UiO : Department of Informatics
University of Oslo

Creating a Framework for Application of Transferability Approach

Robert Kolner

Master's Thesis Spring 2014

Creating a Framework for Application of Transferability Approach

Robert Kolner

May 1, 2014

Abstract

Evolving robot controllers in simulators has been proved to be an effective method of replacing tests on real robots. Flexibility of evolutionary algorithms coupled with cost-effectiveness of running simulations rather than using expensive hardware, made all the mentioned topics an attractive research goal. However, simulating a robot leads to a so-called reality gap, reducing overall viability of the method for creating gaits for quadruped robots. Transferability Approach has been proposed as a method for avoiding solutions leading to reality gap, thus improving performance of the evolved gaits in the reality.

This thesis focuses on creating a framework for conducting experiments with Transferability Approach. A fixed 3D model of Aracna is used in a simulator based on Nvidia's Physx platform as a foundation for creating parameterized controllers. It also features an abstracted model of the real robot, which is used to transfer movements from created gaits to a real robot. On top of that, a surrogate model present in the simulator is able to use data gathered from previous transfer experiments to estimate how well a given gait will perform in the reality.

The framework has been implemented and tested on the real robot. The first results look promising with regard to simulating the robot and transferring gaits to the hardware, despite a considerable reality gap. Transferability Approach was applied without any success, though the amount of data used and generated in the experiment is not large enough to be conclusive. Further investigation is required in order to determine viability of the proposed method.

Contents

List of Figures

Preface

I would like to thank Kyrre Harald Glette, my supervisor and probably the most patient and understanding person at the University of Oslo, for the support and advice you have given me thorough the project. To you, along with Eivind Samuelsen, who did a great job at creating, modifying and introducing me to many of the tools I eventually ended up using: thank you, you have made this paper possible at all!

I would also like to send my appreciations to a fellow student, Tønnes Nygaard, for all the interesting discussions and ideas you have given me. Your hard work and amazing results were a great inspiration.

Lastly, I would love to thank my parents and my family for support. Without it, I doubt I'd ever complete the project and this paper. A special thanks goes out to Jules, my significant other, whose patience and help was invaluable.

Thorough all of this, all of you who've heard out me and my outrageous ideas, stories and excessive amount of not always well-deserved rant: thank you so much for keeping me sane!

Part I

Introduction

Chapter 1

Introduction

This chapter contains an overview of the goals of this thesis, along with the motivation for writing it. In the end, it gives a short outline of the text.

1.1 Motivation

Robotics is a fairly young discipline that combines advanced mathematics, mechanics, electronics and computer science into one with a goal of creating automatic machines capable of executing complex actions. The idea and the dream of such machines has for a long time been present in imaginations of countless people, but it was not until the rise of electronics and computers that the idea could become anything more than a dream. The term robotics has evolved to encompass a lot of different specialized fields, each with its own applications. To give a few examples of possible areas in which robots can be used: they are found in industry [2], medicine [53], nursing homes [34], or even playing football [24].

An exciting field in robotics is developing gaits for legged robots [58, 65, 29, 48], as they enable to create nature-like bipedal or quadruped autonomous robots people always have been dreaming of. Creating gaits is a fairly difficult optimization problem, to which evolutionary algorithms, a concept which in itself was inspired by the nature, have been proved to be among the most promising approaches [6, 49]. Advances with regard to processing power and versatility and creativity of evolutionary algorithms allows them even to be applied to the creation and improvement of morphologies of robots, spawning a field called *evolutionary robotics* [45, 7, 50].

The nature of evolutionary algorithms requires that a lot of testing is done and many attempts fail before what can be considered as a successful solution emerges. This has lead to development of software simulating environments and physics of robots, which allowed bigger populations and longer evolution runs, thus making it possible to increase the amount of parameters and complexity of the evolved controllers and

morphologies. Now that both the creation and testing of the individuals was done inside the computers and not on real robots, a new important problem arose: the solutions created in the virtual world did not behave in the same manner in reality. If developing robots in simulations is to be used for anything more than being a curiosity, it is crucial that the difference is small.

A lot of different concepts were created to reduce the gap between the worlds. One of the newest and most promising ones is a concept called Transferability Approach [36]. If applied correctly, it should be able to remove solutions that are expected to not behave similarly in the simulator and reality, thus improving only the individuals that should do well when transferred.

1.2 Goals

The ultimate goal of this thesis is to investigate how Transferability Approach can be used to reduce reality gap of a fixed, but mechanically complex robot. Before this goal can even be attempted, a robot has to be chosen and a framework for testing created. Therefore, the real goal of this project is twofold:

1. Develop a framework for simulation and real-life experiments on Aracna [39].

2. Extend it to incorporate an environment for conducting experiments using Transferability Approach.

According to [36] Transferability Approach should improve how well solutions generated in simulator transfer to the real world, by identifying inaccuracies of the simulator using earlier experiments on a real robot, and using this information to limit fitness of individuals that are not expected to perform well in reality. This should lead to an overall improvement of performance of gaits generated in simulator on the real robot, and should be easily visible in direct comparison against simulation runs without using Transferability Approach.

A secondary goal of this thesis is to evaluate viability of Aracna as a platform for the process. A good hardware framework in this context should behave predictably in reality compared to the simulation and have clearly defined behaviours which either affect or are affected by performance of a gait.

1.3 Outline

The thesis consists of 4 major parts - introduction, background, implementation and results. Each part is then further split into more specific chap-

ters.

Part 2, the background, summarizes the theory used in this thesis, both for Transferability Approach, its alternatives, description of the used robot and a short description of related work that has already been done.

Part 3 describes how necessary concepts were implemented and why choices and trade-offs made were done. It also gives a brief overview over the experiment.

In the last part the results of the work are discussed and a conclusion presented.

Part II

Background

Chapter 2

Background

This chapter will briefly describe the theories behind the most important concepts used in this thesis.

2.1 Multi-Objective Optimization

A recurring problem in robotics, and computing in general, is that of optimizing a set of parameters in order to achieve a desired goal. Some problems are easier than others, for examplegrid-based pathfinding, where there is a multitude of algorithms able to find the shortest path in a reasonable time [14, 15, 25]. On the other end of the spectrum, there is a set of problems which are believed to be impossible to solve in polynomial time, NP-hard problems, which are described in for instance [59]. Good examples of NP-hard problems include Travelling Salesman or Knapsack Problem, neither of which has known algorithms for solving in polynomial time, even though polynomial-time approximations exist. Common to many of those problems is that they have a clearly defined goal: a pathing algorithm tries to minimize distance. A solver for the Knapsack Problem will try to maximize value while holding weight under a given threshold. Ultimately, there is almost always an optimal solution which the algorithms try to reach.

This is often not how things function in real life. Let us imagine we are driving a car and want to find the best way from home to the airport. There is a path that is calculated to be the best based on the distance and speed limits. In an ideal world that would be enough, but there are many other factors one has to think about when designing a path. For instance - is there heavy traffic? Are there any construction sotes or any other obstacles on the way? Does it snow heavily in any areas overlapping the chosen road? All those factors may or may not affect the viability of the route. Instead of trying to assess exactly how much time we're going to lose due to each of the obstacles, one could define an additional goal for the optimization algorithm - to avoid areas with likelihood of slowing down -

and try to optimize the path with regard to both goals. This is an example of what is called a multi-objective optimization.

Characteristic for this type of optimization is that there is often not an optimal solution we can find, but rather multiple combinations that are be deemed to be more or less equally good, and it is often up to the user to choose which trade-off is the best one. A set of solutions found to be optimal is called Pareto-optimal or a *Pareto-front*, which, while defined differently (see [9], chapter 1.2), often overlaps the former [1]. An optimal solution in this context is one that is not dominated by any other. We say that a data point (p_1) is dominated by an other (p_2) if both following points hold true:

1. No objective from p_2 is worse than p_1: $\forall i(f_i(p_2) >= f_i(p_1))$
2. There is an objective in p_2 that is better than corresponding objective from p_1: $\exists i(f_i(p_2) > f_i(p_1))$

All non-dominated data points are the result of the optimization. So, looking back at the traffic example, we would get the shortest route somewhere in the Pareto front, but we would also have the route that is the most free of obstacles in the same set. Additionally we would get many points that lie somewhere in between.

This thesis uses Evolutionary Algorithms, which are among the most popular algorithms for solving multi-objective optimization problems.

2.2 Multi-Objective Evolutionary Algorithms (MOEA)

In the last section, we were using multiple objectives to choose the most optimal solutions from a given set. We didn't say anything about how the set is developed. Before we can go further, let's shortly describe a few simple concepts related to evolutionary algorithms.

Individual

Single candidate solution to a given problem produced by an Evolutionary Algorithm (EA). It contains a *genotype*, which in turn affects individual's *phenotype*. A genotype is a vector containing genes, each of which might be expressed as one of many data types, including bits, integers, floating point numbers and strings. This thesis is going to use a vector containing 32-bit floating point numbers as a genotype of a solution, and to keep things short and concise only operations relevant to this type will be described.

[1] By the formal definition, Pareto-optimal set is a set of points which are optimal with regard to the whole search space, while Pareto-front is the set of current, non-dominated solutions

Population

A set of individuals in a generation. The size of the population used for a run in EA is a trade-off, and its size often depends on search space and available computational resources. Too small population numbers can lead to little exploration and hence bigger chance of becoming stuck in local maxima, which means that globally better solutions might not be found at all in a run. On the contrary, increased population size beyond a certain point, which depends on the size of the search space, will not improve performance of the EA and is thus merely wasting resources. There is no simple way to assess with certainty beforehand how big the optimal size could be without making assumptions which might or might not hold [27], and it needs to be adjusted based on empirical data.

Generation

Evolutionary algorithms are iterative, and therefore contain a concept of generations. At every point in iteration, a population is generated based on the previous generation. Individuals from earlier generations, *parents*, are used to generate a new set of individuals, so called *offspring*. The amount of generations in a run that is required for convergence is not constant and, just as population size, often has to be decided empirically. In single-objective scenarios it is possible to terminate a run once a satisfying solution is found, but with MOEA we often want to get a Pareto-optimal set of solutions to choose from. As such, we can either set the amount of generations to be constant or add a condition that terminates the run if there is no improvement in the last n generations.

Exploration and exploitation of search space are important and often conflicting concepts in EAs. If we imagine search space as a landscape, where higher ground represents better fitness, and value of latitude and longitude are values of a genome vector with 2 elements, then exploration represents how big part of the landscape is surveyed, and exploitation is how well the EA climbs a hill or mountain once it finds a slope. Being stuck on a small hill represents a local maximum in this scenario. Though rate and balance between exploration and exploitation can be adjusted by using different operators and their parameters, the exact relation itself is not obvious [11]. Neither are the requirements of a problem with regard to the balance between

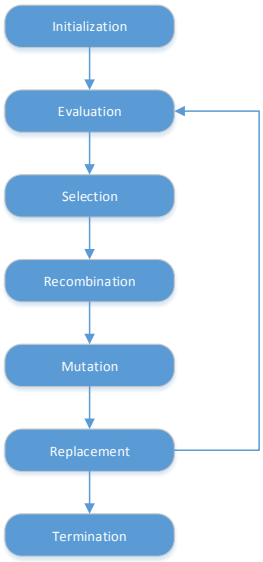

Figure 2.1: Structure of evolutionary algorithms

them: parameters giving a good solutions to
one problem might be suboptimal for another
one. There are many different approaches to the problem of tuning parameters, with most common being (all from [11]) trial-and-error, following general guidelines, using paremeterless EA, past experiences, identifying features of the fitness landscape, statistical analysis of control parameter interactions and their effect, mathematical models, and meta-evolution[2].

2.2.1 Evolution

The process of evolution contains multiple steps that can be seen in figure 2.1. This section will briefly describe what every step does.

Initialization
An initial population is often generated randomly, though knowledge about the problem or fitness landscape might be used to make a better initial set.

Evaluation
Fitness values are calculated. Alternatively, the population is sorted based on relative fitness between individuals, if it is not possible to compute an exact score.

Selection
Evolution is based on the survival of the fittest, therefore the individuals ranked higher get more spots for themselves and their offspring in the next generation. There are many different strategies for selection, with the most popular, according to [43], being ordinal-based[3]. There are multiple ordinal-based selection algorithms in use, like tournament selection [42], (μ, λ) selection (also known as *comma-selection* [30]) and truncation selection [12].

Recombination
A crossover operator is applied to two or more [16] individuals with a goal of creating offspring which takes positive behaviours from its parents. One-point, arithmetic recombination [17], n-point, and uniform [52] crossover, all give different results with regard to exploration and exploitation.

Mutation
In mutation phase, genes of an individual from the set of offspring

[2] In other words: looking at choice of evolution parameters as an optimization problem in itself

[3] Based on relative ranking within a population instead of their fitness compared to sum of fitness in population. An alternative is proportionate-based and includes techniques like proportionate selection, stochastic remainder selection and stochastic universal selection [43]

are randomly changed using a mutation operator. For floating-point representation of genes, it is normal to adjust the value by adding a randomly generated number from a fixed distribution [17]. Uniform and Gaussian distributions are the most common to use, with exact parameters depending on the range of the genes.

Replacement

In this phase a part of the population is replaced with the set of offspring. In it's simplest form, all parents are replaced by their offspring, which might in the end lead to less efficient hill-climbing and loss of the best solution. To avoid the problem, a concept of steady-state, or elitist recombination has been created [54]. Quite similar in concept, but working for a whole population is a concept of elitism where several of the best individuals are guaranteed to not be replaced for next generation. Both algorithms are described in chapter 3.3 in [40].

After the replacement phase the evolutionary algorithm either goes back to the evaluation-step, or terminates the process and returns the last population as a result.

2.2.2 NSGA-II

MOEAs have inherently two potentially conflicting objectives: the distance to the Pareto-optimal front should be minimized while diversity between individuals is maximized. Different strategies have been devised to deal with the problem, for example NSGA [41], PEAS [35], SPEA and SPEA-II [64] and others [28, 20]. The framework used in this thesis, ParadisEO [3], uses the second revision of Non-dominated Sorting Genetic Algorithm, or in short NSGA-II, as described by [13]. The goal of the algorithm is to fix problems of other approaches, such as high computational complexity, lack of elitism and need for specifying parameters. NSGA-II is based on an effective implementation of sorting populations by level of Pareto-optimality, which means finding Pareto-front, moving the points to an own, level-1 set, finding another Pareto-front based on remaining points, moving them to a higher level set, and so on, until the population have been exhausted. A naive implementation of the sorting algorithm is in the worst case $O\left(MN^3\right)$, with number of objectives M and population N [13], but the proposed algorithm reduces its complexity to $O\left(MN^2\right)$. In the context of the last section, the sorting of population based on Pareto-optimality is the first part of evaluation phase, and is paramount to movement towards the Pareto-optimal front.

In order to satisfy the second objective of MOEAs, diversity between individuals, NSGA-II uses the concept of estimating density by calculating crowding distance of individuals in Pareto-fronts. A crowded-comparison

operator is then devised based on rank and crowding distances, and used to decide a winner in subsequent tournament selection.

2.3 Reality Gap

Simulators are not able to represent every aspect of the real world. A complex simulator that attempts to represent all known physical phenomena also has the disadvantage of being computationally expensive, to the point of being slower than testing in the real world. In some situations the dynamics of a robot can't even be described, because they are not fully understood. There is, however, no reason to simulate everything, as for each use case there is only a set of features that are needed for a fairly accurate representation of the model and its dynamics. For instance, if we're developing gaits for a simple robot, there is no reason to incorporate complex calculations involving fluid dynamics just to compute air resistance. Or, perhaps more realistically when developing the same robot, we won't be able to perfectly simulate the forces in place between its legs and the ground. By approximating or abstracting away some of the factors we are making the simulation less complex, and thus faster. One of the drawbacks is that the approximated phenomena can become too simple and affect how behaviours are performed, so that we can end up with a difference between what we see in the simulation and what actually happens in the real world. Once a difference is big enough to be observed from a point of either an observer evaluating behaviours or the robot controller, we can talk about reality gap. It is inherently an undesired phenomenon, because it decreases precision, and therefore the usefulness of a simulator. If a simulation is precise enough to give the same outcome as when a behaviour is performed in reality, we can talk about crossing reality gap.

The following sections will briefly describe a few alternatives to deal with the reality gap.

2.3.1 Minimal simulation

The fact that a perfect simulator is neither practical nor possible, poses a question that must be answered before we can go any further. What does a simulator have to be able to do in order to be able to cross the reality gap? We have already made a distinction between the robot controller and its environment. Only the latter of the two has to be simulated, the controller is by definition an entirely digital and virtual entity.

So, in order to answer the question, let us start from the beginning: how do these two separate parts interact with each other? We have an environment with a set of rules governing what the entities within can and cannot do, and we have a controller with a limited set of actions it can

perform. Let's say we have a room with no air in it and an aeroplane-shaped robot with a propeller standing on the ground. The robot can perform one action - fly - and has one sensor - an altimeter. In this scenario the robot will not be able to do anything. The only action it can do, the only *behaviour* it can exhibit, is to fly using lift generated by air, while the environment does not support this behaviour. A behaviour in general is defined by how an object interacts with and changes its environment from the point of view of an observer, rather than a controller's. For instance, if our airless room was an inside of a Zeppelin filled with a void, the robot might perceive it and act as if it flies, based on its readings from the altimeter, but this would not be an example of it performing a behaviour. It does not matter that the controller produces the correct output for flying by giving power to the engines and steering the ailerons. The state of the environment does not change as if the robot was flying because of robot's actions. However if our Zeppelin was be filled with gas, the propellers would create a force that would push the robot forward and possibly cause it to fly.

Note that for our purposes the environment does not *actually* have to change in any particular way, it just has to interact with the controller as if it was. Let's say that the Zeppelin is airless again, but this time its control system is connected to our controller in a way that causes it to change altitude in the same manner as the little aeroplane would with the same controller output. For our purposes the robot flies as long as the Zeppelin behaves in the same way as the plane inside it would.

For every such behaviour, there is a set of features that the environment has to support in order for it to be possible. The set of all those features is called the base set. An environment that supports the base set of a behaviour will also support the behaviour. Following this logic we can deduce that a dynamical system can be modelled correctly if and only if the environment supports all behaviours of the robot.

So, why aren't simulators perfect?

Even if we define a complete set of features a simulator must have in order to be able to model an environment that supports all necessary behaviours, it does not mean it will actually cross the reality gap. Every feature might either be possible to simulate precisely enough or not, but not one of them can actually be simulated with 100 per cent precision. Another interesting thought is that there might be features which when simulated, may hinder evolution and lower fitness of the robot in reality due to their unpredictability. [31] deals with this problem by subjectively judging fitness of how well a feature transfers to reality based on how reliably it performs in reality a particular behaviour evolved in the simulator.

With this in mind, we may conclude that a controller can reliably the cross reality gap if it is uses a set of features that can reliably cross the

reality gap by themselves. But that conclusion is simply not true. As mentioned before, a simulator, however complex, can never simulate *any* feature entirely correctly, due to the enormous amount of dependencies and interactions in the real world. Fortunately, crossing the reality gap does not mean that an individual has to perform in the exact same manner in simulations and in reality, but merely that it is good enough. It is, however, still advantageous for a controller to perform reliably despite some inaccuracies in simulation.

All of this leads us to the definition of minimal simulations. To quote [31]:

> [Minimal simulation is] the simplest type of simulation capable of evolving controllers for real robots. It starts by examining the minimal set of features that a simulation must include if the performance of a particular behaviour within that simulation is to be possible in the first place. It then goes on to examine the minimal relationships that these features must bear to reality if transfer across the reality gap is also to be possible.

Now, the minimal simulations have a set of both advantages and trade-offs. The advantages are implied, but there are also drawbacks of this minimalistic approach.

In fact there is one main disadvantage. By limiting the base set, we're also severely limiting possible routes for evolution. While this might lead to fewer unusable solutions, it will also lead to fewer *usable* solutions and decrease chance for any evolution. Limiting evolution also deprives it of few of its main advantages - creativity and adaptability. Minimal simulations work well for cases where those advantages are not needed, but only then.

2.3.2 Back to Reality

Back to Reality (BTR) attempts to solve one of the difficulties of making a simulator - adjustments of parameters. As mentioned before, we can divide a robot system into two components - robot controller and environment. In this case, as in all other cases, admittedly, there are two distinct environments, reality and simulation. What an organism perceives about its surroundings is all that it can know about, and with the correct framework it is not supposed to be able to detect whether it is running in a simulation or not. This means that for the purpose of a robot controller there is no difference between the environments, which means that any controller can be transferred freely between them.

One of core concepts in BTR is that the robot learns by alternating between reality and simulation. Studies suggest that doing so might by itself improve the performance of the resulting solution [23]. According

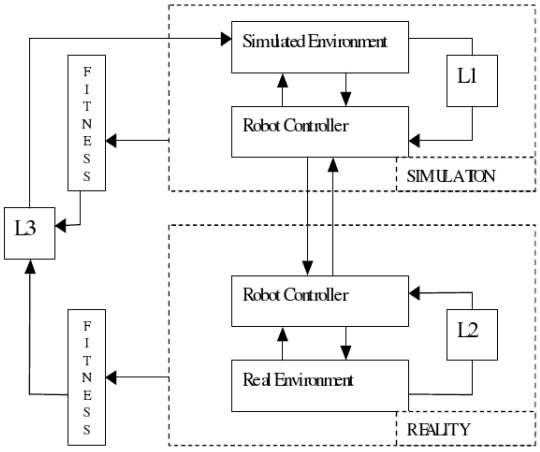

Figure 2.2: Back to reality building blocks – [61]

to [61], learning by alternation will only work if reality gap is decreasing, which can only happen if the simulator is continuously improved as there is more data on how it differs from reality. The last point brings us to the main point of BTR - in addition to applying learning algorithms to robot controller, it does so with simulator. BTR consists of three main, sequential stages - evolving the controller in simulator, evolving the controller in reality, and eventually comparing results and evolving the simulator. Figure 2.2 illustrates roughly how BTR is built up.

Exactly which learning algorithms are used for each phase is not specified. However there are some choices that might perform better than alternatives. Due to a limited amount of data, it is usual to prefer reinforced learning algorithms for developing the controller in reality (L2). At the same time we can obtain a lot of data from simulations, so genetic algorithms might be used for developing the controller there (L1). There is even more freedom when it comes to evolving the simulator (L3), as the data is scarce and the parameters are many, and both reinforced learning and genetic algorithms are viable alternatives.

2.4 The concept of transferability

The reality gap sometimes occurs due to a simulator finding "shortcuts" of a kind; behaviours which considerably improve robot's performance in the virtual world, but are impossible in real life. As an example of such a situation, let us imagine we are trying to optimize a controller for increased movement speed of a walking robot. In the process the simulator, by randomly applying excessive force to robot's joints, or possibly because of not simulating gravity, mass or inertia correctly, would make the robot jump, thus increasing its fitness to levels unattainable in reality. This behaviour is obviously undesirable and should be avoided.

There is a concept developed to minimize the reality gap called *Transferability Approach*. Instead of attempting to improve the simulator, it approaches the problem from a different side: it makes the assumption that however accurate a simulator is, there will always be at least minor discrepancies between what is happening in virtual and real worlds. Transferability approach aims at finding out which parts of the simulator represent reality well, and which do not, and avoids solutions that do not transfer well to the real world. The concept is thus based on an assumption that transferable parts are represented in viable solutions and not in solutions where the reality gap is larger. [36] defines a concept of transferability as an attribute of a controller:

> A controller is said transferable if the corresponding behaviours of the robot observed in simulation and in reality are similar.

Controllers that exploit the inaccuracies of a simulator have lower transferability than the ones that do not. Based on this fact alone a conclusion can be made that transferability and efficiency in simulations are not necessarily compatible goals. In cases where they are, there is no problem and we have no reason to do anything more. Otherwise [36] proposes using a Pareto-based Multi-Objective Evolutionary Algorithm to optimize the solution with two goals in mind: how well a controller performs in a simulator and how well it transforms to a real robot.

A measure of how well an individual transfers from simulation to reality - in other words transferability - is called disparity value. More distinct solutions (with greater distance between simulation and reality) usually have greater disparity value, but to be precise, disparity is a measure of the distance of behaviours. For a description of what a behaviour is, look at section 2.4.1. We assume that if behaviours are well-defined, then any distance between simulated and real worlds, when using the same controller, is always the result of a reality gap. Therefore we can use disparity value as an objective in simulation, decreasing priority of less transferable individuals. We can also assume that there is a threshold over which the disparity value is too high for a solution to be suited for transfer into reality. It can be used to avoid solutions which obviously cannot be transferred and are as such not a good solution to the given problem. The exact value of the threshold has to be determined empirically.

In order to obtain the disparity value, we define and use so-called STR (simulation-to-reality) disparity function D^*. The result of the function corresponds directly to the discrepancy in behaviours exhibited in simulator and reality. For any controller $c \in \mathfrak{C}$ and behaviour $b(c) \in \mathfrak{B}$, the exact disparity value can thus be defined as $D^*(b(c))$. The function is initially unknown in all search space, so we have no way of knowing the value before conducting transfer experiment. Getting a precise number for

every solution would be highly impractical and would defeat the whole purpose of simulating in the first place, so we have to find a way to estimate it.

In both scientific and engineering problems, we often use surrogate models to approximate results of operations that otherwise would be too complex to run efficiently or would take too much time ([33], [10], [62], [37]). Their role is to use already gathered data about the search space to interpolate to unknown data points. In case of the Transferability Approach, we can build a surrogate model to approximate the disparity function based on its value for other inputs. In the beginning we do not have any data we can use to build a reliable model, or any model for that matter, so some transfer experiments have to be performed. There are a few constrains that have to apply to make this process viable (from [36]):

1. The number of experiments has to remain small

2. Close behaviours in simulation should have close STR disparity value

3. The experiments are iteratively or periodically generated

Ideally, we want to be able to approximate a wide range of possible solutions. Extrapolation is known to have higher uncertainty than interpolation, so the greater part of the behavioural space $\mathfrak{B}$ is covered, the greater the accuracy of the approximation should be. In other words, we should initially transfer gaits as distinct as possible from each other in terms of behaviours. We can measure how different two controllers c_1 and c_2 are using equation 2.3. If we assume we already have a set of transferred controllers $\mathfrak{C}_t$, we can calculate diversity of a not transferred controller c with regards to the set:

$$diversity(c) = \min_{c_i \in \mathfrak{C}_T} b_{dist}(c_i, c) \tag{2.1}$$

So from a set of available controllers $\mathfrak{C}_P$, we can choose the next candidate for a transfer experiment by maximizing the diversity function. If $\mathfrak{C}_T$ was defined beforehand, then we could stop here and just choose the next candidates as necessary. We do, however, begin with an empty set. The goal is to have a set such that:

$$\mathfrak{C}_t = \max_{\mathfrak{C}_n \in \mathfrak{P}(\mathfrak{C}_P), |\mathfrak{C}_n| = N} \sum c \in \mathfrak{C}_n \min_{c_i \in \mathfrak{C}_n} b_{dist}(c, c_i) \tag{2.2}$$

This gives us an optimal set with N elements designed for a transfer experiment. The problem is shown to be NP-hard ([57]), but can be solved in polynomial time given a few restrictions which will be described in 3.4.4.

2.4.1 Definition of behaviours

It is important to explain what is meant by close behaviours, or rather how behaviours can be compared at all. Each behaviour is described by n values, called behavioural features. Behavioural features are a description of a given behaviour and should be chosen in such way that any pair of behaviours should have distinct vectors, with similar vectors producing similar disparity values. Exact definition depends on details of the problem. For instance [36] used three different measures in their experiment: distance covered, mean height of the geometric centre of the robot thorough the experiment and its orientation at the end. Once the features are computed, we can obtain a value for behavioural distance between two individuals. Let's say we have two controllers that we want to compare, c_1 and c_2. They exhibit distinct behaviours, described by behavioural features vectors b_1 and b_2. Behavioural distance between c_1 and c_2 is the given by

$$b_{dist}(c_1, c_2) = \|b_1 - b_2\| \tag{2.3}$$

From here on, a surrogate model can be created using one of many approximation techniques.

Once the model is created, it can be used to estimate the transferability of defined behaviours. Initial samples may not be enough to make a reliable model, so the dataset has to be updated with new points as they becomes available. This problem will be described later, in chapter 3.

2.5 Aracna

The robot used in this thesis is called Aracna. It is an open-source hardware solution designed by [39], based on their previous experience with evolutionary robots. The main design goal of Aracna is to fix all of the shortcomings of the previous iterations, the main point of which was the excessive weight of its legs due to servos, which caused the servos to not be able to provide enough power in the innermost joints to execute the commands given by the controller. In order to get rid of the problem, all servos have been moved to the body of the robot and connected to the legs using cranks. This decision resulted in a considerably shifted weight balance: each leg of Aracna weights 105g, compared to the roughly same-sized QuadraTot, with its leg-weight of 217g. It has also reduced the angle range of the leg joints - with servos right at the joints, the joints themselves could move around almost full circle. The nature of the mechanical system in Aracna limits the arc of the inner joints to 28°-49°, and the outer joints to 63°-103°. This has the practical implication of simplifying simulation and experiments in real life, due to smaller search space and inability to do any self-damaging movement on a flat ground.

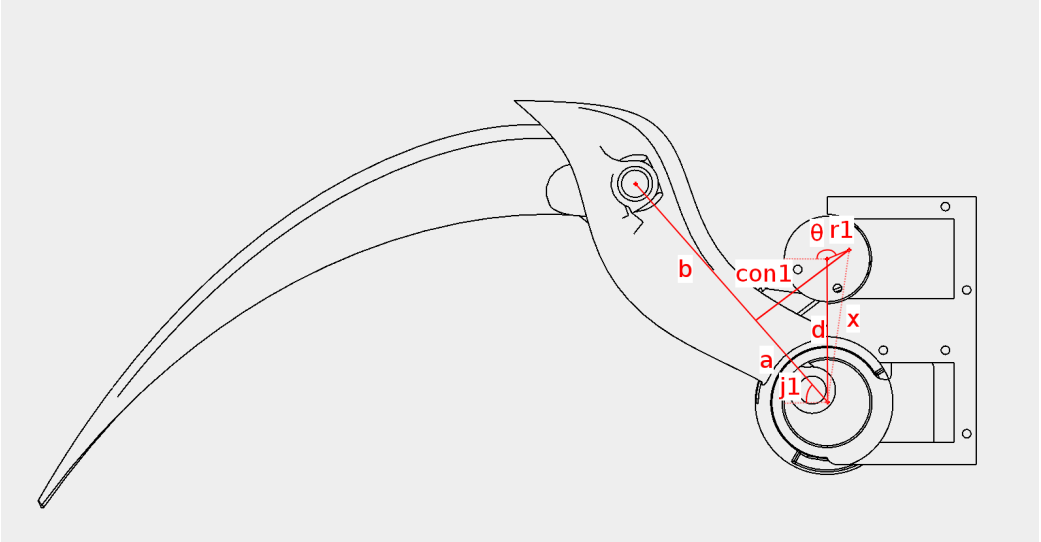

Figure 2.3: Sideview of a leg with annotation for inner joint.

part	length (mm)
d	39.0
a	28.0
b	47.0
con1	30.0
r1	5.1

Table 2.1: Dimensions necessary for computing angle j_1

Additionally, such solution might provide a more interesting, if not better base for evolutionary algorithms. According to [39] this is an area where evolutionary algorithms should perform better than human engineers in generating well-performing gaits.

A major difference from QuadraTot and similar robots is how the movement of the servos translate to the angles of the joints. Instead of directly controlling the anglular positions, the servos transfer force using beams. In addition to limiting the angles, it allows the servos to work in continuous free rotation mode. This means that the controller can

part	length (mm)
a	28.0
b	47.0
c	16.0
con2	80.0
r2	5.0

Table 2.2: Dimensions necessary for computing angle j_2

Dimensions	$32mm \times 50mm \times 40mm$
Weight	$54.5g$
Voltage	$9\ 12V$
Angular resolution	0.29°
No load speed	$97rpm$
Stall torque	$1.8Nm$

Table 2.3: Technical specifications of Dynamixel AX-18A

use either one of two possibilities - it can either steer the joints using a sequence of positions, or by set the velocity of the motors. In its simplest form the controller of Aracna needs just to set angular velocity and the initial phase of the 8 servos to constant values to be able to generate a multitude of repetitive gaits. The angle for the inward links in legs with regards to the horizontal plane can then be, if necessary, computed using following equations:

$$x^2 = d^2 + r_1^2 - 2dr \times cos(\theta_1 + \frac{\pi}{2}) \tag{2.4}$$

$$j_1 = \frac{\pi}{2} - \cos^{-1}(\frac{d^2 + x^2 - r_1^2}{2dx}) - \cos^{-1}(\frac{b^2 + x^2 - con_1^2}{2bx}) \tag{2.5}$$

The j_2 angle for the outer joint can be computed in an analogous way, with an exception of an addition of a constant value due to the curve of the outermost links in the legs.

$$y^2 = b^2 + r_2^2 - 2br \times cos(j_1 - \theta_2) \tag{2.6}$$

$$j_2 = \pi - \cos^{-1}(\frac{y^2 + b^2 - r_2^2}{2yb}) - \cos^{-1}(\frac{y^2 + c^2 - con_2^2}{2yc}) + 0.3687 \tag{2.7}$$

2.5.1 AX-18A

Aracna uses Dynamixel AX-18A servos to control the angular position of its joints. The model is an upgrade from the previous AX-12A, providing higher maximum speed and more torque. The technical specifications for the servos can be found in [1], and a short summary of the most relevant characteristics can be found in table 2.5.1.

AX-18A contains a micro-controller which can communicate with other devices using a serial connector. The micro-controller can be used to control both positional attributes (position, movement velocity), mechanical limits (limits on angle, torque, temperature, voltage), attributes of the controller (compliance slope, punch) and to read current status, position, and velocity. All the information above is discrete in the controller, most often with a range from 0 to 1023, with the highest value corresponding to limitations of the servo.

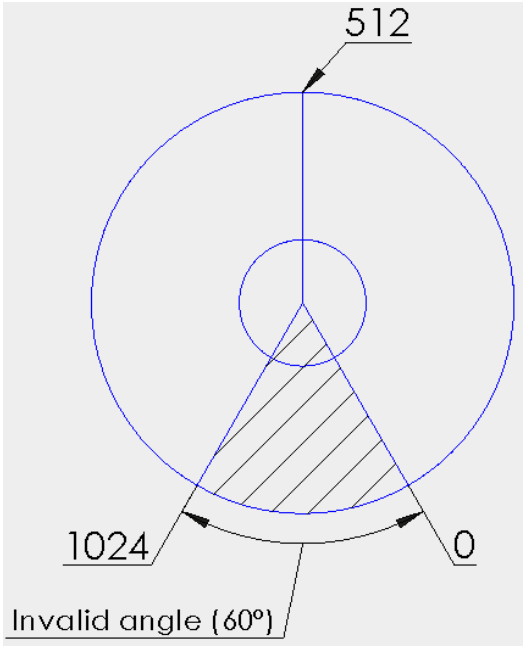

Figure 2.4: Positional values in AX-18

The controller has two different modes of operation - it can either set position of the servo (joint mode) or its velocity (free rotation mode). In joint mode, position is limited to range $[0, 1023]$, which does not actually cover a whole rotation (fig. 2.5.1). AX-18A has a zone about 60° wide which cannot be reached with positional control. Thus, $[0, 1023]$ in the controller is linearly mapped to $[0^\circ, 300^\circ]$, or, in radians, $[0, \frac{5\pi}{6}]$, in the physical servo. This limitation can be avoided by using free rotation mode, which is achieved by setting both minimum and maximum angle limits in the controller to 0 and steering the servo by using moving speed parameter. However, while the servo is going through the zone which is unavailable in joint mode, the values for current position returned by servo are either 0, 1023 or random.

Both modes use a common set of parameters:

CW/CCW min/max angle limits In joint mode, they describe the minimum and maximum position (clockwise (CW)/counter-clockwise (CCW)) to which the servo can be set, with range $[0, 1023]$. In they're both set to 0, the controller enters free rotation mode.

CW/CCW compliance slopes sets amount of torque to use when the servo's current position is near its goal position. Range is $[0, 255]$, with each bit corresponding to one step and only the most significant bit that is not zero being used. Valid only in joint mode, does nothing in free rotation mode.

Moving speed describes how fast the servo is moving to its goal position.

One unit corresponds to about $0.111rpm$, and the range of the parameter is $[0, 1023]$. In free rotation mode moving speed says how fast the servo is moving as a percent of maximum velocity, but, differently from the joint mode, the range is $[0, 2047]$. Range $[0, 1023]$ is used for movement speed in the counter-clockwise direction and $[1024, 2047]$ is used in the clockwise. In short - the most significant bit indicates direction, the 10 last bits indicate velocity.

Max torque Limits how much torque, as in percent of the maximum, the controller is going to apply to the servo. The limits on maximum torque are $[0, 1023]$.

Part III

The experiment

Chapter 3

Implementation

This chapter will give insight in how the described background was used to create a platform for simulating and testing Aracna. An overview over general structure of the whole platform can seen in figure 3.1.

The chapter will also inform about any discrepancies from theory and challenges encountered during the process of implementation.

3.1 Simulator

The experiments used a custom simulator developed for internal use by the ROBIN group at University of Oslo. It is based on Nvidia's PhysX-framework for simulating physics and ParadisEO, which is a framework for genetic algorithms and multi-objective optimization [3]. Initially, the simulator was used to generate robots, both morphology and controllers, by the means of evolution. The results were promising, but to simplify the problem in this thesis and avoid any additional inaccuracies, a robot with an already defined body was chosen. Between all available ones, Aracna [39] was selected as the platform for the experiments. More information about the structure of the robot can be found in section 2.5.

Thus, before anything else, the evolution of the hardware had to be turned off and a model of Aracna implemented. The existing implementation of a model in the simulator was more general than the scope of this project required, so it was relatively simple to recode it to use another model. Creation of the model itself presented a few challenges.

First and foremost, the physical skeleton had to be created. Simplicity was prioritized due to limited resources and faster simulation times, so while the model of the robot is structurally correct, it doesn't simulate a lot of details; most importantly, it doesn't simulate the intricate joint control system of the Aracna. Instead of simulating the movement of servos and applying force to joints through a connector, as in the physical robot, the simulator controls the links directly, only mapping simulated position of the motors to calculated angular position of the joints. This

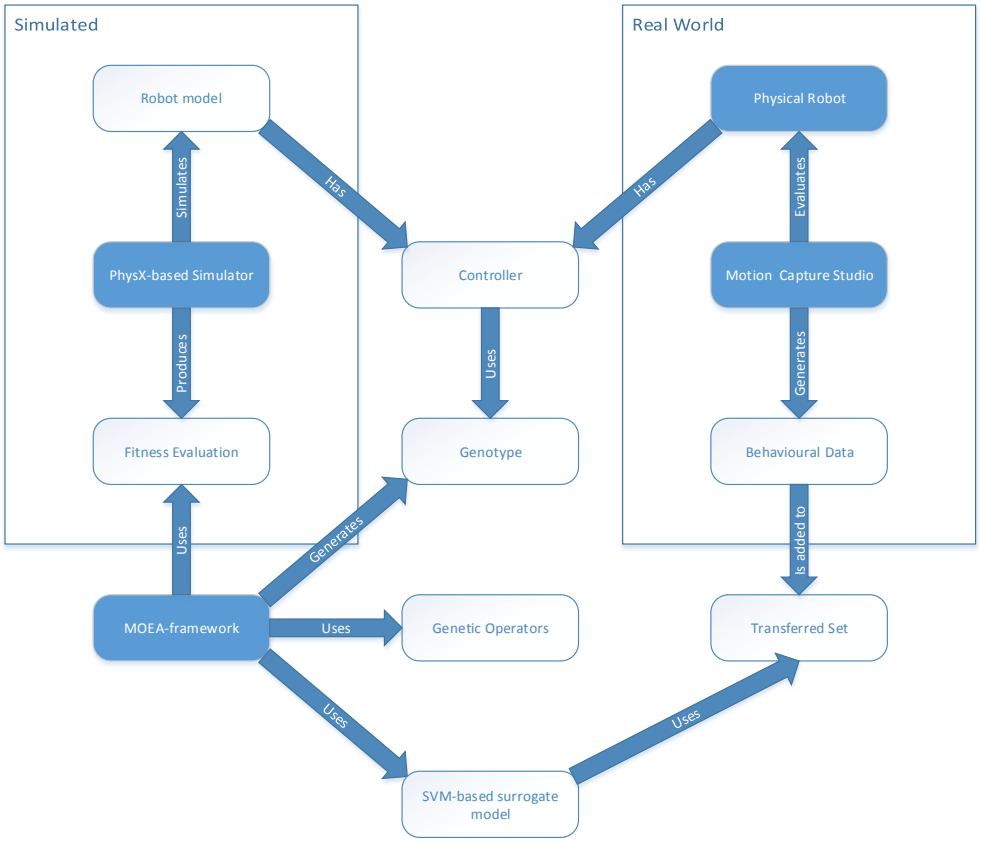

Figure 3.1: Structure of the implementation used for this thesis. In white - parts of the implementation written and/or added in this thesis. In blue - unchanged parts of the preexisting code

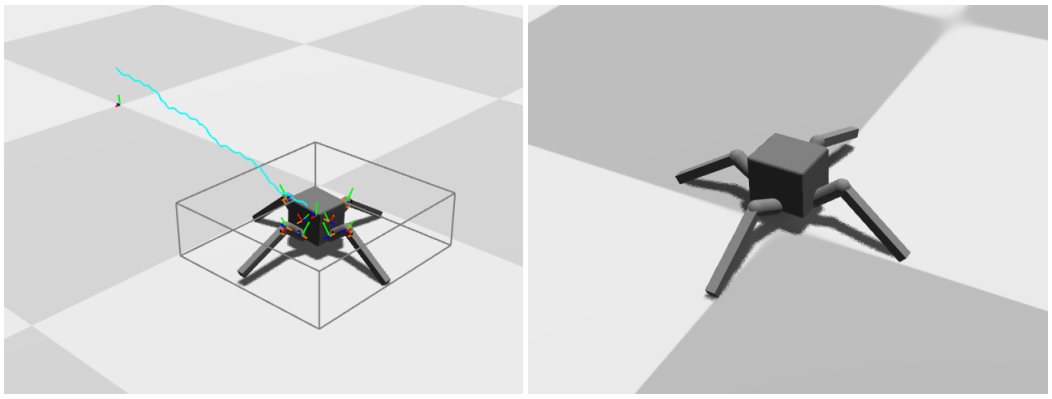

Figure 3.2: Simple model of Aracna in simulator with and without its bounding box and path

has the effect of it not being able to easily calculate the correct power and maximal torque the servos should have in the simulator. The problem is is ignored by assuming that we always have enough torque in reality. Another difference from the real robot is the contact area between the tips of the legs and the surface the robot is standing on. The dynamics are quite difficult to simulate due to their form and various surfaces, and the values of friction in simulator had to be chosen through empirical evidence.

The structure of the robot itself is vastly simplified. As shown in figure 3.2, the central body is just a rectangular box (which actually resembles its real counterpart quite well) with dimensions $102mm \times 102mm \times 117.5mm$ and a mass of $870g$ plus $8 \times 54.5g$ for the mass of the servos. From the block the four legs are placed in appropriate positions and orientations. Each leg is composed of two parts - an upper capsule-shaped object, with a length of 75mm, the radius equalling 15mm and a weight of 65g, and connected to the main body in one end and a cuboid, which represents the outer part of the leg on the other. The cuboids are sized at 170mm x 10mm x 10mm, with a mass of 40g. All given dimensions are roughly equal to the real Aracna.

3.1.1 Evaluation

The evolutionary algorithm used in the simulator allows us to easily create gaits that maximize fitness based on either single or multiple given objectives. A description of how evolution itself works is given in section 3.2, the focus of this subsection is on which objectives were considered and chosen for determining the fitness of the individuals.

Distance evaluator is the most basic and most obvious objective for the evaluation of movement. While simple, the calculation of the fitness value can be done in multiple ways, based on which direction we want the robot

to take. The already implemented default objective calculated the distance in z-axis from the starting point and ignored the other axes (fig. 3.3). The gaits evolved using this objective had all to move in the same direction as the leg containing servos with id's 0 and 1, limiting the amount of viable gaits.

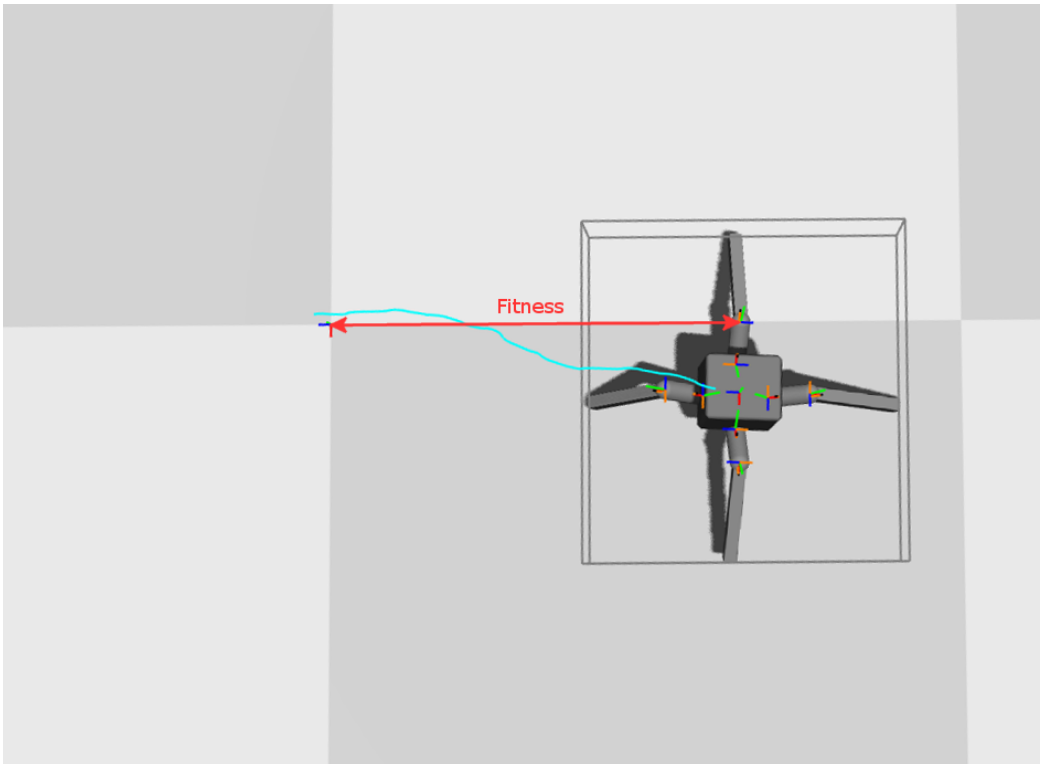

Figure 3.3: Default implementation of the movement objective based on z-axis

Aracna is symmetrical in 4 directions, which means that gaits in North, East, South and West directions are basically the same, with phases rotated by $\frac{\pi}{2}$. To evolve another family of gaits, we will have to change direction to North-East, North-West, South-East or South-West. This can be achieved either by rotating model of the robot by $\frac{pi}{4}$ around y-axis, or by defining a new goal, the latter being considerably simpler. Manhattan distance got its name from how distance is calculated on the 4-connected grid of streets in New York: total distance is a simple sum of x- and z-axis. Assuming gaits in all directions are potentially equally viable, solutions scoring high on both axes will dominate ones scoring high in one of them, with the consequence of the robot prefering to move in the NE direction. The fitnesses of gaits evaluated using this method are not directly comparable to ones obtained using the other mentioned methods, because they do not actually represent a real distance. In addition to fitness an additional measurement of the movement distance in the simulations had to be taken, which is what the graph 3.6 at the end of this section shows.

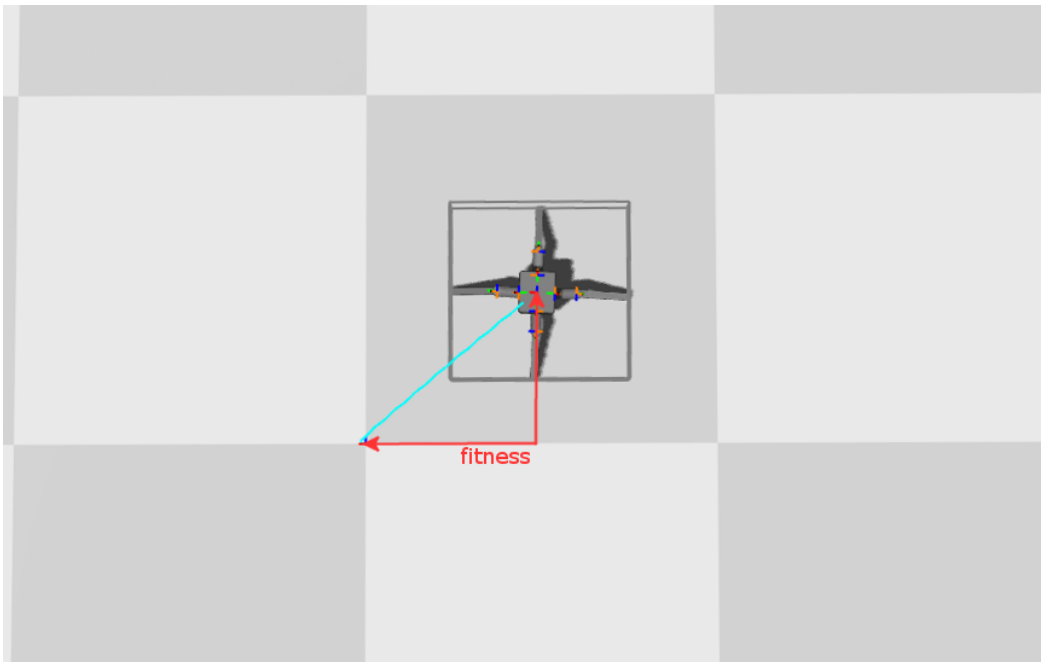

Figure 3.4: Using Manhattan-distance as a movement evaluator

Another possibility is to use calculate the distance from the start point using the Pythagorean Theorem. $\sqrt{x^2 + z^2}$ gives us absolute displacement, disregarding any information about direction of the movement. This method of evaluating gaits gives us the greatest search space, with all types of motion having an equal and fair chance. As such, once we find an acceptable solution, we can redefine which way the robot is supposed to go instead of trying to force it in a specific direction. The main drawback is a greater search space, which leads to slower (less directed) evolution and thus worse gaits found in the same time. Another minus is that direction of movement is not set, so any comparisons of orientation or direction of movement as a part of computing disparity values are impossible, or at least not as simple. This can be avoided by limiting positive score to positive values in x and z axes.

In the end every one of them was implemented and compared based on the distances achieved in the simulator. Two runs were done for for each type of evaluation, with 64 individuals and 1024 generations each. Few of the best individuals from each run were taken and tried on the real robot. Based on the results from the experiment, with the data from the best runs shown in figure 3.6, coupled with the fact that it provides the greatest search space, the Euclidean evaluation of the distance was chosen as an objective in further evolutionary runs. Greater search space is normally not an advantage, but in this case gaits generated with restrictions of movement direction were far to similar to be discernible using Transferability Approach.

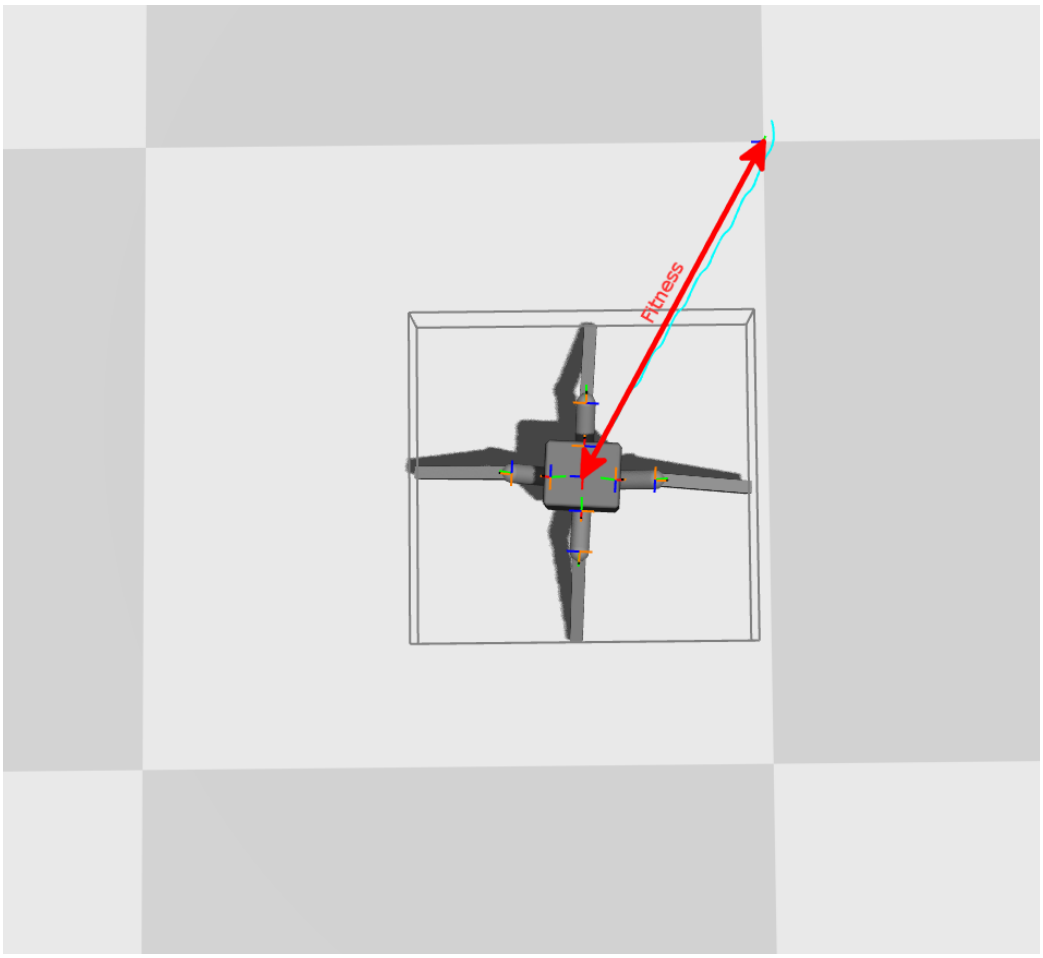

Figure 3.5: Movement objective using displacement from start point

Turn evaluators Both left and right turn evaluators give a value describing how much body of a robot has turned in 8s. They are useful for evolving gaits for turning 180° around, often for use in testing in real environment, where the space is limited and the robot has to be either turned or moved manually to have enough room to test a gait. They can also be used to ensure a gait is straight, or at least that orientation of the body doesn't change, which combined with the periodicity of gaits shorter than 8s should mean that the gait is more or less straight. The symmetry of Aracna means that even if the orientation stays the same, the direction of the movement can change to either direction. That is also one of the main reasons for why the turn evaluation was not used in any of experiments in this thesis.

Transferability As an evaluation parameter, transferability is supposed to discern how well a controller should behave on the real robot. The evaluation happens at the end of a single simulation of a gait and uses

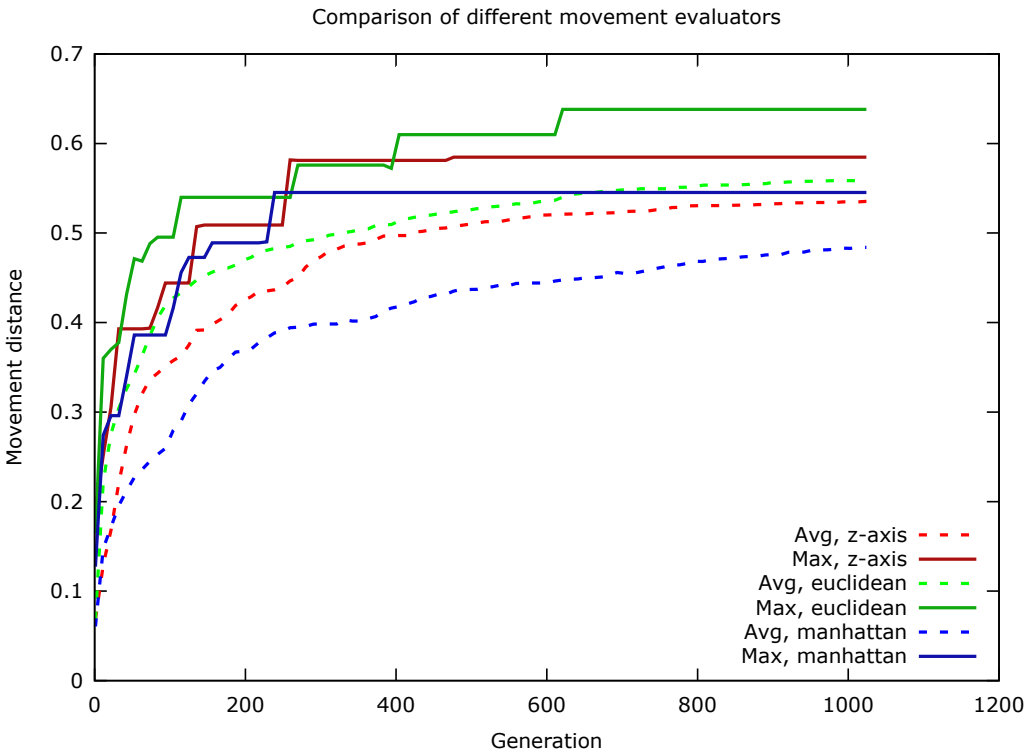

Figure 3.6: Comparison of fitness over time in chosen runs using different movement evaluators

data gathered during the gait. There are a lot of different measures that can be used in the evaluation (see 3.4.3). This data is compared against a library of already transferred gaits using LibSVM.

3.1.2 Genotype

Evolutionary algorithms improve fitness by manipulating genes of individuals in a semi-random fashion. Thus the first step in creating an evolvable controller is to define its genotype and how the genotype is mapped to phenotype. Aracna has 8 servos, one for each of the 2 joins in one of the 4 legs. Usually, in similar robots, joint angle is linearly codependent on the angle of the servo, but as already mentioned this does not apply to Aracna. In order to centralize weight and reduce mass of the legs, Aracna contains all its motors in its central piece, with bars going out to respective links and controlling their pitch (see section 2.5 on page 20 for more details). One of the consequences of the system is that the motors can be controlled in two different ways. Controlling by specifying a sequence of positions over time may give us more complicated, irregular gaits. One could argue that a bigger search space can be an advantage, but in order to simplify the experiment as much as possible, the other option was chosen: specifying a constant velocity and phase for each servo. This also makes defining and implementing a genotype simple. Two parameters per motor were used, in total 16 parameters for whole robot. All genes are encoded as 32-bits floating point numbers. Velocity of a servo i can have values between $v_i \in [-1, 1]$, where both limits represent maximum attainable speed in different directions. Phase p_i ranges between $p_i \in [0, 1]$, with linear mapping $[0, 1] \rightarrow [0, 2\pi]$. Position of servo i with regard to time can then at all times be obtained using a simple equation:

$$P_i(t) = (2\pi \times (v_i \times t + p_i)) \bmod 2\pi \tag{3.1}$$

In practice, the AX-18 servos use discrete values for controlling of both position ($[0, 1023]$) and velocity ($[-1023, 1023]$). This means that some precision will be lost when transferring data to the joints. The simulator was adjusted to incorporate same limits for precision, so the reality gap would not be in any way affected. This also underlines another problem, which will require us to limit precision anyway.

All generated gaits are regular, but depending on exact values for velocity, some have longer period than others and, as mentioned before, the simulations are limited to 8s. With this limitation we assume that 8s is enough to decide viability of a gait. However, solutions with period longer than 8s would not be tested completely, so results of the simulations involving them would be at best inaccurate, at worst completely wrong (fig. 3.7). This shouldn't necessarily be a problem as the gaits can after all be restarted each 8s, so whatever happens after that time is irrelevant.

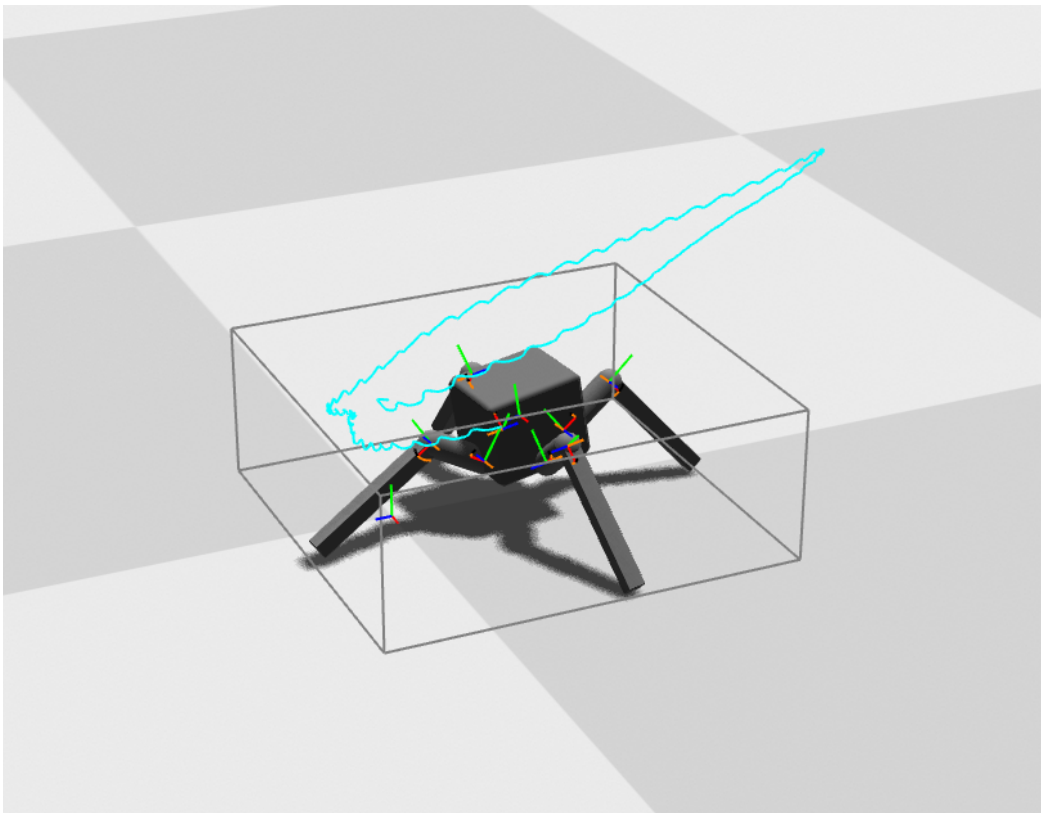

Figure 3.7: An example of a gait with a period longer than 8s

This is a good enough reason for testing if adding a limitation to velocity in order to ensure that period is shorter than 8s affects performance. The restriction was added in the following way. Output velocity for the servos can be set to 1024 different values for each direction, so in the worst case scenario, we would have one servo's velocity set to $v = 1023$, and another one to $v = 1024$. This would mean that if they're both starting with $p = 0$, then they'll have to make respectively 1023 and 1024 rotations before they once again both have exactly $p = 0$ at the same time. With servos performing at $97rpm$, the gait would have a period of roughly 10.5 minutes, which is obviously unacceptable. On the other hand, if we restrict values to strictly -1, 0 and 1, they will rotate with constant relative phases and period of about $0.62s$. Thus, at maximum speed, each servo can make $97\frac{rot}{min} \times \frac{8s}{60\frac{s}{min}}] \approx 12.93rot$. Rounding down, we get a maximum of 12 rotations per 8s, which gives us 12 different velocity levels in each direction.

Adding velocity levels proved to not affect the perfomance of gaits in the conducted simulations. Therefore, the restriction was used in the experiment.

crossover	avg fitness	max fitness
1-point	0.22659	0.334384
none	0.25563	0.346968
arithmetic	0.15366	0.188951

Table 3.1: Average and maximum fitnesses achieved in chosen early runs with different crossover operators, population=64, generations=128

Evolution operators The next step in the process is defining the operators which will work on the genotype when creating offspring for a new generation. After testing a few different crossover algorithms, a conclusion was made that none of them improve either how fast good solutions are produced or how good the best individuals become in the end. Quite the contrary - on the average the results were slightly better with no crossover operator 3.1. Though the amount of individuals and generations in the runs were small, the results were indicative of the crossover, at least one 1-point and arithmetic recombination operators, *not* improving score or time of covergence. With that in mind, the decision was made to not use any crossover operators at all.

Gaussian Mutation [26] is the operator that was chosen for mutation. It works by going through each gene in the genotype and adjusting its value by a random floating-point number $N(0, \sigma)$, where σ is a parameter describing rate of mutation. Different values for σ change the rate at which individuals mutate, affecting how fast the algorithm improves the average and maximum fitnesses, how often it gets stuck in local maxima, and how well it climbs the hill once it finds a good solution.

A situation where value of a gene goes out of allowed range due to mutation is solved by capping it at the limit of the range. To show by example - if value of σ_v becomes lower than -1 due to mutation, its value is set to -1. Similarly, if its value becomes higher than 1, it is set to 1.

3.2 Evolution Parameters

As mentioned in section 3.1, we have in total 16 different parameters in the genome. It is quite a big amount for an evolutionary algorithm to optimize, but there is no simple way of reducing the amount without changing the basic design of the controller. This should mainly affect the optimal size of the population, but not much more. If we cannot make any assumptions about an optimization problem, we cannot expect one algorithm to perform better than another [27]. That implies that there are no exact answers to what parameters of an optimization algorithm should be, including, in this case, how big the population should be, and what parameters should be given to SVM when estimating transferability.

We do however know that there are many parameters to optimize, and

for that reason experiments should be run with a rather large population to be able to exploit bigger part of the search space. Amount of generations is set to be constant in the simulator, though of own choosing. After initial tests and adjustments, parameters with which all simulations should be run were chosen to be 64 individuals and 1024 generations. This ensures a big enough initial spread of solutions and enough time for convergence.

Another value that had to be found empirically was mutation rate. After experimenting with different values, standard deviation was set to $\sigma_v = 0.2$ for velocity-related genes, and $\sigma_p = 0.1$ for phase. The main reason for why σ_v is twice as big is the greater range of allowed values - $\sigma_v \in [-1, 1]$ versus $\sigma_p \in [0, 1]$.

3.3 Controller in hardware

Developing a controller for simulator is only half of the job. The same controller must behave similarly in reality if we want to get any grounds for comparison. There are 2 options to choose between: steering servos by velocity or by position. By controlling rotation using velocity, we avoid all the problems of AX18 (described in 2.5.1), because we never have to read the state of the servos. In practice, we could set velocity to a constant value and start rotation after a delay dependent on the evolved phase and velocity. There is, however, a real danger of a huge, non-linear and unfixable reality gap, as the motors would get out of sync depending on the resistance. Using position to control the motors is somewhat counter-intuitive in this case, as the core design of Aracna suggests free rotation of the servos, but might be necessary to avoid the mentioned problem. The disadvantage is that reading the state of the motor might return bogus values for the dead cone ($300^\circ \sim 360^\circ$), which would lead to misbehaving in an unpredictable manner for as long as the servo's angular position is in the cone. The proposed solution is to bypass the problem by never positioning the servos inside the cones. Due to the nature of the Aracna, for motors i:

$$\forall i(f(\alpha_i + \theta_i) = f(\alpha_i - \theta_i)) \tag{3.2}$$

Where $f : \Re \to \Re$ is a function mapping from servo angle to its connected joint angle, α is the angle between $\theta = 0$ and the position of maximum retraction and θ_i is the angle of servo i. That means that to get all possible joint positions, we only need to use $\hat{\theta}_i \in [0, \pi)$ or $\hat{\theta}_i \in [\pi, 2\pi)$, calculating $\hat{\theta}_i = 2\alpha_i - \theta_i$ when $\alpha_i + \pi > \theta_i > \alpha_i$.

The robot has 8 servos, each of which has a dead zone in a different place. As mentioned before, angles of the servos are linearly mapped from $[0, \frac{300}{360} * 2\pi)$ to $[0, 1023]$. $\theta = 0$ is defined as the position at which motors have their output splines placed furthest away from the body of the robot, $\frac{\pi}{2}$ with regard to the base plane. We need to know the offset

from motor's 0-point to the defined 0-point. We also need to know what α is when the leg is fully retracted and from it decide what interval to choose. The interval will have to have a length of 614[1], which corresponds almost exactly to π radians. We do not know anything about positioning of the servos beforehand, so all of those will have to be measured manually. Therefore the offsets and alphas might not be precise. This is another factor which might increase the reality gap. All content of the following table is in units used by the servos, i.e. from 0 to 1023.

id	offset	α	interval
0	190	650	$(36, 650]$
1	465	0	$[0, 614)$
2	548	1023	$(409, 1023]$
3	522	0	$[0, 614)$
4	818	0	$[0, 614)$
5	512	0	$[0, 614)$
6	520	1016	$(402, 1016]$
7	793	280	$[280, 894)$

The only limitation with regards to the dead zones is that neither the points of full retraction nor the points of full extension can lie within it. If they do, we either cannot use this method, or have to physically adjust the initial position of the servos on the robot. Using this table the mapping function 1 can be developed.

The pseudo-code will produce a response similar to Figure 3.8. This approach is not without its own problems: it is possible for the algorithm to produce incorrect results in certain situations. If position adjustments would happen continually, the response would be exactly the same as with velocity-controlled motion. However, this is not the case in reality. The controller sets the output for servos at an interval depending on speed of the computer on which it is run. Let's look at the figure 3.8 again and assume that the last desired position was 800. A lot of time has gone since the last adjustment; the next desired computed position is 600. Normally the servo would go through whole motion, with physical position going all the way from around 600, up to about 700 and back down to 600. In my case, the servo will do nothing, as it already is in the physical position we want it to be. Hence, the longer the interval between the updates, the less precise this approximation will be near the edges of the limited movement range. This interval is, in my case, about 17ms, so it shouldn't have any impact on the physical results.

[1] $\approx \frac{1023 * \frac{360}{300}}{2}$

input : Desired *position*, servo *offset* from the 0-point and the point of full retraction α
output: Position with regards to physical servo position

if $\alpha >= \pi$ **then**
  $position := 2\pi - position;$
end
$position := (position + offset) \bmod (2\pi);$
if $\alpha < \pi$ **then**
  $low\_interval := \alpha;$
  $high\_interval := \alpha + \pi;$
else
  $low\_interval := \alpha - \pi;$
  $high\_interval := \alpha;$
end
if $low\_interval > position$ *or* $high\_interval < position$ **then**
  $position := (2\alpha - position) \bmod (2\pi)$
end
return *position*

Algorithm 1: Mapping from desired position to physical position depending on a servo.

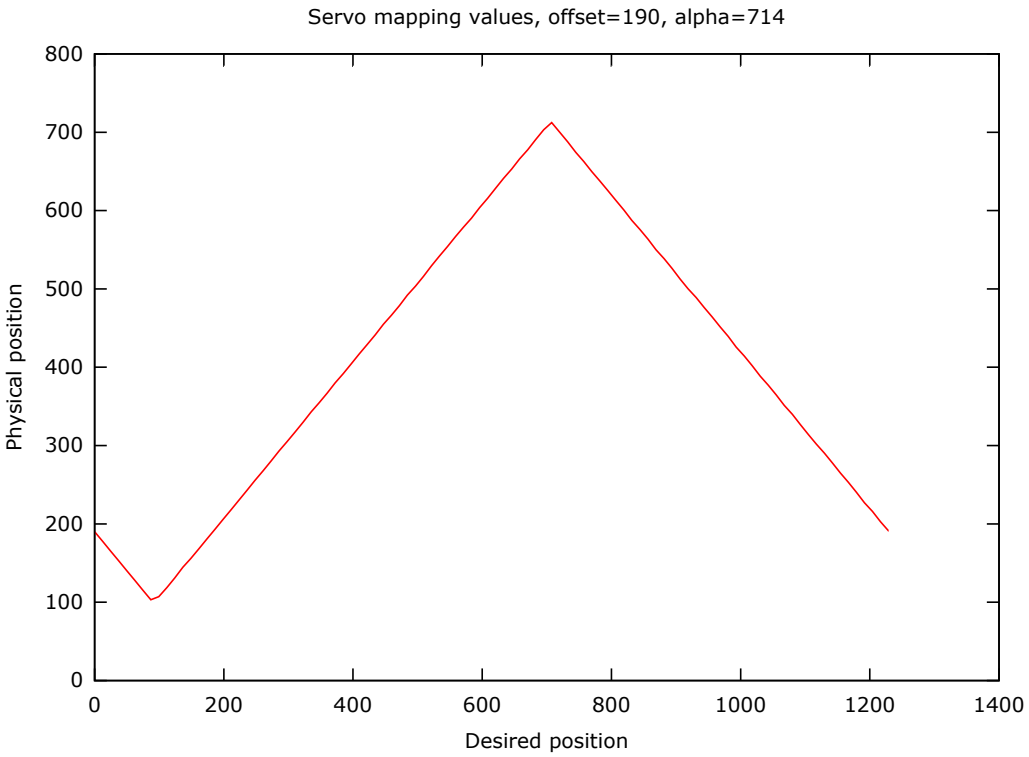

Figure 3.8: Relation between desired and physical position of the servos for example values.

3.3.1 Going back to reality

The output of the controller is an angle based on current time and the mentioned internal parameters. In the simulator, this is enough to be able to move the legs into the desired positions. The reality is more tricky. In the simplest case, we can order a servo to move to a desired position every time we compute a new one, in which case the speed of the rotation has to be set beforehand. This will lead to servo moving in the direction which will lead to least movement. A constant angular velocity implies that if the desired rotation is further away than the servo moves at given speed, the real movement might be off phase, as it lags behind the calculated angle. An obvious solution to this problem is to set the velocity to maximum, in which case the servos can reach their goal before the new position is given, which will lead to a very stuttering motion. This effect is obviously undesirable. Thus, velocity will have to be adjusted depending on angular distance between the desired and real states.

To solve the problem, a simple PID-controller was implemented, as described in Chapter 3 in [46]. The total equation for output of the servo in velocity units is:

$$v_i = K * (\epsilon_i + \frac{1}{Td * D_i} + Ti * I_i) \tag{3.3}$$

Where v_i is output velocity of servo i, ϵ_i is error os servo i:

$$\epsilon_i = target_i - state_i \tag{3.4}$$

D_i is the derivative part of PID:

$$D_i = \frac{d\epsilon_i}{dt} \tag{3.5}$$

And I_i is the integral part of the PID:

$$I_i = \int_0^t \epsilon_i \mathrm{d}t \tag{3.6}$$

With correct gain-constants, there should be a small, non-noticeable lag. The main advantage is a smooth movement of the servo, small delays and fast response to changes. On the opposite side, this adds another layer of complexity and a place where correct choice of parameters is paramount for obtaining good results. After experimenting with different gaits and angular velocities, the values given in table 3.2 were used. The size of K parameter is due to the difference in which we measure error - amount of radians - and the unit in which we define velocity, with a range of $[0, 1023]$.

K	600.0
T_i	6.0
T_d	3.0

Table 3.2: Gain-values for PID controller

3.4 Transferability

3.4.1 Problems and challenges

Firstly, I should mention that my implementation of Transferability Approach is not fully compatible with the one described by [36]. An important point in their article is defining a threshold for diversity, τ. After evaluation of each generation, all generated individuals have their behavioural features used to compute distance from the set of already transferred controllers using the equation for diversity (2.1). Every individual with diversity greater than τ is tested on a real robot, and the results are put back into the set of transferred controllers. This ensures that there are at all times grounds for estimating the disparity function for existing set of controllers.

Due to the pre-existing code structure of the simulator, changes required to implement this feature would be difficult. Fortunately, the first simulations indicated that the amount of different possible gaits for Aracna is quite limited. This means that given a big enough initial set of transferred solutions, lack of continuous transfer experiments should not affect the results as much as it could have. So while I would have liked to implement it, I had to reckon that it is infeasible to do in the amount of time I had.

3.4.2 Surrogate model

The first choice that had to be made was of regression method. [44] mentions Artificial Neural Networks (ANN), Inverse Distance Weighing (IDW) and Support Vector Machines (SVM) as viable possibilities. In my experience SVM tend to perform well with small data sets, which is the case here. Additionally, some experiments ([60]) have shown that SVMs have slightly better accuracy than IDW. In the end, as this choice doesn't make that much of a difference for all practical purposes, the solution with a better C++ library was chosen, as it considerably shortens development time and possibility of human mistakes. The library used in this project is LibSVM ([4]).

3.4.3 Behavioural features

Successful application of Transferability Approach requires a good choice of parameters as input to its model. There are arguably many different descriptors that can be used for this, but as amount of training data is low, we will have to choose them carefully. Some of the attributes that can be used for our goals are:

- Distance moved
- Direction of the movement
- Orientation at the end of the movement
- Total amount of height and/or orientation changes
- Minimal/Mean/Maximal height of the centre of mass of the robot
- Maximal speed of the centre of mass of the robot

Note that to be able to apply the disparity function, we have to be able to measure the chosen parameters in both simulator and real world. Due to the nature of the real world testing framework, the only parameters we're able to obtain is position and orientation of the robot at any given time. Of those two, only position can be considered as reliable: even after extensive calibration of cameras used in the studio, the application used for computation of position and orientation lost sight of the markers and calculated entirely wrong orientation based on the remaining ones. This happened every few seconds and lasted few frames. The position returned by the software was mostly unaffected by this problem. In order to be able to use orientation data anyway it had to be free of any very sudden changes. This was done by making a simple filter which compared current orientation with last valid orientation, calculated how many radians per second the necessary rotation would have, and throwing away all data with more than 2 rad/s.

3.4.4 Initial transfer set

Given a set of results from simulation, we need to find few individuals for a transfer experiment. As mentioned in 2.4, amount of gaits chosen for the experiment should be rather small, and as diverse as possible. Equation 2.2 gives us an optimal set, but is an NP-hard problem, so it had to be simplified by adding a few restrictions. The problem becomes much simpler if we give it target population size N instead of trying to find an optimal subset from the whole powerset. The set of possible solutions is reduced from $\mathfrak{P}(\mathfrak{C}_P)$ to $\mathfrak{P}_N(\mathfrak{C}_P)$, which in turn reduces amount of possible solutions from:

$$2^{|\mathfrak{C}_P|} = \sum_{i=0}^{|\mathfrak{C}_P|} \binom{|\mathfrak{C}_P|}{i} \tag{3.7}$$

To:

$$\binom{|\mathfrak{C}_P|}{N} \tag{3.8}$$

A limit on population is also necessary because we want to limit the amount of transfer experiments.

In order to make it even easier, we manually define an initial point from which the rest of the population would be chosen one-by-one. This is so that all the next individuals after the first would be the ones with greatest diversity of the remaining points. Sets obtained using this algorithm are *not* optimal. To compensate, in order to get good enough initial data, chosen population size N was be slightly larger.

Chapter 4

Validation and Experimentation

This chapter will present the configuration of the testing environment and benchmarks used for evaluating of performance. Some of the conducted experiments have already been described in chapter 3, as a part of defining parameters and algorithms. Here, I will present the experiments used to evaluate Aracna as a platform and Transferability Approach as a mean of reducing reality gap.

Validation-part is twofold - into simulator and hardware part. Correctness of simulation and evolution of gaits is the topic of the first part, section 4.1. Once we've ensured that the software part does its tasks, we have to validate that the generated controllers behave similarly in both simulation and reality. This is done in section 4.2. In the end, in section 4.3, we will design a short experiment to check if the simple implementation of Transferability Approach used in this thesis reduces reality gap.

4.1 Simulator

The first step in implementation was to create a simulator able to generate viable controllers. While not necessarily part of the thesis' goals by itself, it is important to ensure that controllers perform well, are repeatable, i.e. they do not fail in runs longer than 8s, and that their parameters correspond to the motion seen on the screen.

The settings for the evolution were already given in the chapter 3, but to rehash it quickly, the algorithm is based on NSGA-II with following parameters:

crossover	none
mutation	Gaussian, $\sigma = \{0.2, 0.1\}$
population	64
generations	1024
objectives	max movement (euclidean)

6 runs will be run with these parameters. From each run, individual with the highest fitness will be chosen and closely examined. There are 3 things that will be examined - distance moved, period of the gait, and genome of the individual. Due to the simple mapping from genotype to phenotype of controllers, the genes can be directly used for comparison with the actual motion.

4.1.1 Evaluation

Distance of movement will be calculated using Euclidean distance in z and x directions. This measure is important for ensuring that generated gaits do anything at all. A gait will be considered as a successful if the robot moves at least $40cm$ over the course of $8s$. The distance is chosen quite arbitrarily, as there are no previously conducted experiments on Aracna with public results. $\frac{40cm}{8s} = 5\frac{cm}{s}$ should guarantee that the movement was not a result of random fluctuation and is in fact a gait.

Period of a gait can be easily calculated based on controller's genotype and servo velocities, but in this section, I'm more concerned with how the gait performs in the first 8s versus the second 8s. All gaits are guaranteed to have a period shorter than 8s, but starting position can have a significant influence on travelled distance in the first 8s. Therefore, for each of the chosen controllers, a 16s simulation will be run, and the distance d_t from the start point is going to be saved at $t_{run} = 8s$ and $t_{control} = 16s$. We can then define a relation:

$$k = \frac{d_{t_{control}}}{d_{t_{control}} - d_{t_{run}}} \tag{4.1}$$

If the value of k differs significantly from 1, then the gait can't be said to be stable. Values significantly less than 1 are not expected to occur.

In the end, I will examine genome of the individuals to spot potential inconsistencies.

4.2 Hardware

Once the simulator is shown to be working, we need to make sure that the controllers transfer well to the hardware. The same chosen controllers from the last section will be tested on the real robot.

The first checkpoint is to set controller in the starting position, i.e. position at $t = 0$, and check if all servos are in the right positions with regard to the genotype. As mentioned before, the genotype contains 16 parameters, 8 denoting velocity and 8 corresponding directly to phases. At $t = 0$ velocity doesn't matter (see equation 3.1) - position of the servos should directly correspond to the genes.

Once we're sure that the servos are in correct position, the model from simulation will be compared with the real robot. There should be no significant differences in the initial position.

The same process should be done for $t = 2$ and $t = 8$, to ensure that the mapping from controller to hardware is correct. Given that there are no significant error at any stage, the gaits on hardware will resemble the simulated ones, bar reality gap.

4.2.1 Evaluation

Evaluation of how well the controllers transfer will be purely visual and approximate. Due to reality gap possible already at this stage, careful manual examination should not prove to be inferior to measuring relevant angles and distances. Any significant inconsistencies will be noticeable; unnoticeable inconsistencies are considered insignificant.

4.3 Reality Gap and Transferability Approach

The last part of the project consists of testing the created gaits in reality, assessing their reality gap and applying the Transferability Approach in order find its effectiveness.

From the 6 runs run in 4.1, 4 will be used for creating the initial transfer set $\mathfrak{C}_T$, as described in 3.4.4), while the remaining two will operate as a control set. The transfer set will consist of 20 individuals, which means 20 initial transfer experiments.

All experiments will be performed in ROBIN's motion capture studio (section **??**). 4 reflective markers were carefully placed on the physical robot as far from each other as possible, while keeping the distance between them unequal and ensuring that they are not on the same plane. Figure 4.1 shows the final placement of the markers. The marker on top was notably important - with the rod just a bit shorter, the motion capture software tended to lose markers and miscalculate position and orientation of the robot, despite extensive calibration. The power cable was mounted on pulley wheels with enough slack to reach almost whole canvas and weights on the other side, so that it's pulled back up when too loose.

Before placing the robot on the canvas, both the cameras and the software had to be calibrated. In the process of calibration, target area has to be covered using a rod with a marker at the end to provide enough data points for the software to conduct necessary calculations[1]. Once calibrated, we can create a model of the robot in the software using its markers. OptiTrack Motive then constantly sends position of the tracked

[1]As the software is closed source, it is not specified what kind of calculations are involved in the process

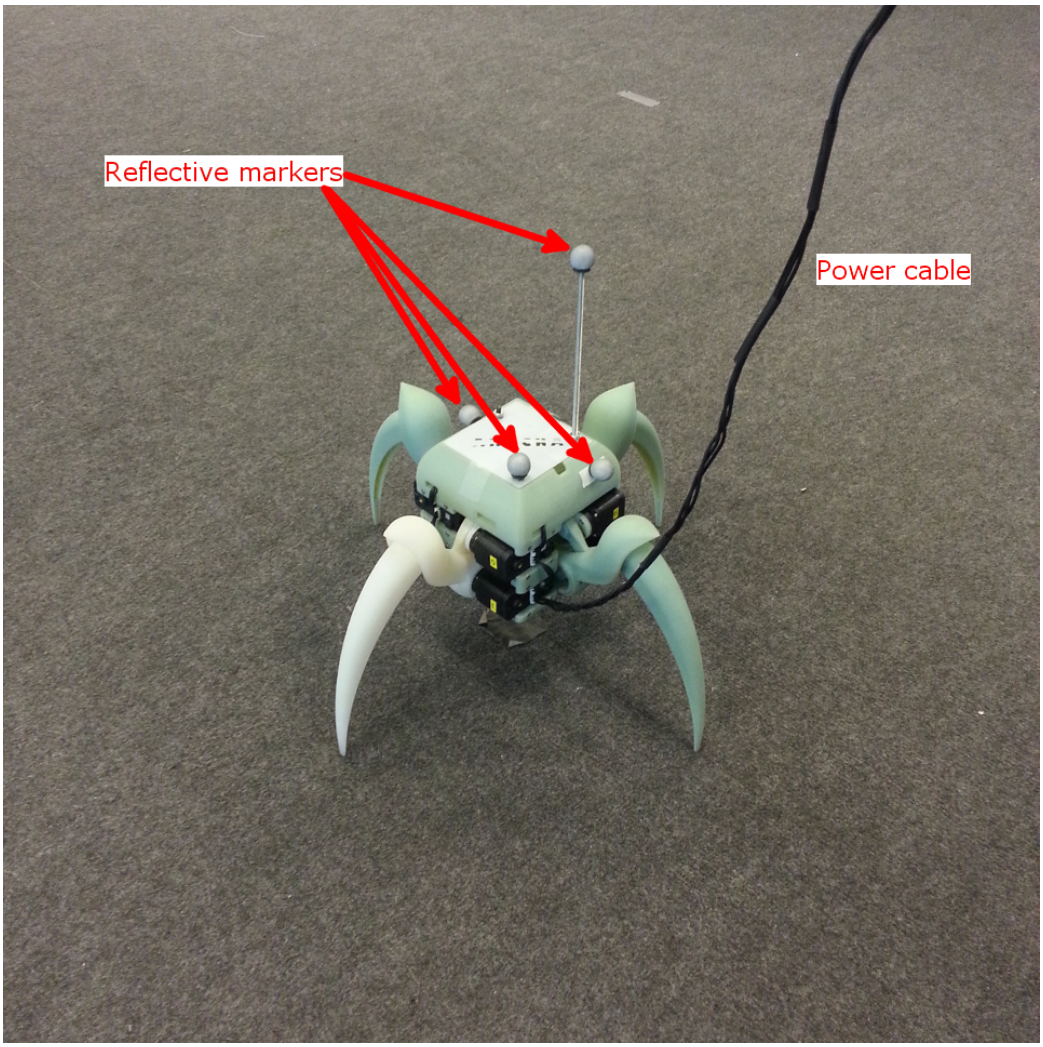

Figure 4.1: Aracna with reflective markers

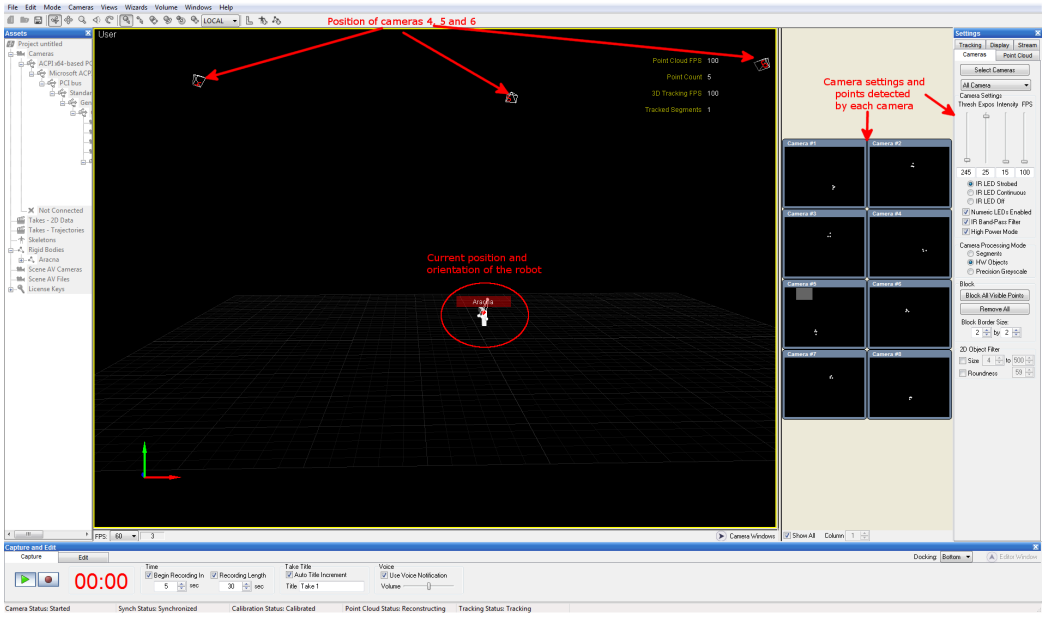

Figure 4.2: OptiTrack Motive software used to track the position and orientation of the robot

object to the simulator, which then decides what to do with them, thus allowing to syncronize times and positions from simulations with times and positions from the real robot.

The rest is done from the simulator - both choosing gaits, controlling the robot, and registering necessary data. Figure 4.3 shows the outline of the testing process. After every run in reality, the robot will be manualy set back to the centre of the testing area, before the next trial is started to attempt to normalize the conditions as much as possible.

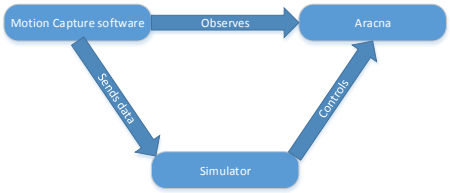

Figure 4.3: Overview over motion capture process

Once behavioural data from reality for the set of individuals is obtained, 2 new simulation runs will be run, this time with two objectives - to maximize movement and minimize the estimated disparity value. Among the resulting solutions, 10 will be chosen for next transfer experiment based on their diversity. In the end, two last simulation runs will be done. From each of the last runs 10 of the best solutions will be chosen to be tested on the real robot and compared against a corresponding amount from the control set, in total 40 attempts on the real robot.

From the list of possible behaviours, I will use following 3:

Direction moved Angle around y-axis in which the centre of mass moved from the start-point. This measure is designed to eliminate solutions that perform just as well, but move in the wrong direction, which can be indicative of it happening as a fluke rather than desired behaviour.

Distance moved Measure of how far (in z- and x-axis) an individual has moved. The behaviour is computed using the same method as the movement objective. Including distance in the disparity value calculation should increase diversity of transferred sets and test both successful and not so successful gaits.

Sum of changes of orientation Total sum of all changes in orientation of the central body during an 8s run. In my test runs I've noticed a big difference between individuals in how much their orientation changed, and how it affected the physical runs. Thus, I assume it might be a good separator for transferability of solutions.

The resulting vector will be normalized with $\left[\frac{1}{2\pi} \quad 1 \quad \frac{1}{8}\right]$.

4.3.1 Evaluation

I suspect that there will be a considerable reality gap in the first set of transfered solutions. Optimally, with regards to further testing, there should be a lot of variability in performance in reality compared to the simulations, as this will create grounds for applying Transferability Approach. Despite its inaccurate usage (3.4.1) I still expect including transferability to improve how well generated gaits transfer. Thus, the reality gap and spread of distances in reality should be smaller in the second iteration of tests.

All 20 individuals from the last run on the real robot will be evaluated and compared based on their movement distance in 8s. Their fitness will also be compared to how they performed in the simulator.

Part IV

Results

Chapter 5

Results

In this chapter I will present results for each of the sections in chapter 4. Every section starts with an overview over obtained results with explanation, which is followed up by an evaluation and short discussion. Comparison of scores and results is done where necessary.

5.1 Simulator

After setting up the simulator and fixing remaining faults in the code, runs nr 58-63 were conducted with settings from the last chapter. From each run, the last generation (1024) was taken, and average and maximal fitness in population calculated. The results were as follows:

run	avg fitness	max fitness
58	0.69107	0.786380
59	0.73567	0.831076
60	0.73937	0.821551
61	0.73740	0.820551
62	0.69988	0.822203
63	0.77244	0.873999

The given runs were all conducted with a single goal - maximizing movement distance - thus fitness is a direct measure of the objective. All values are measured in distance in meters travelled in $8s$.

Another interesting measure is how fitness of populations changed over time. A simple graph can say a lot about choice of parameters for evolution. For each generation in each run, an average and maximum fitness were measured and plotted in figures 5.1, 5.2 and 5.3.

Second part of validating viability of the solutions is determining their repeatability. From each run, an individual with the highest fitness was chosen and tested in the simulator. The results for $t = 8s$ and $t = 16s$, including value k obtained using equation 4.1, are given in the following table:

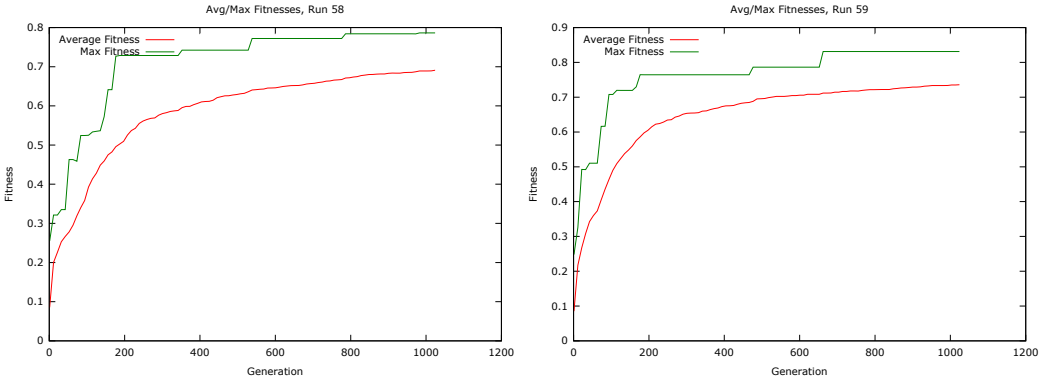

Figure 5.1: Fitness over time, runs 58 and 59

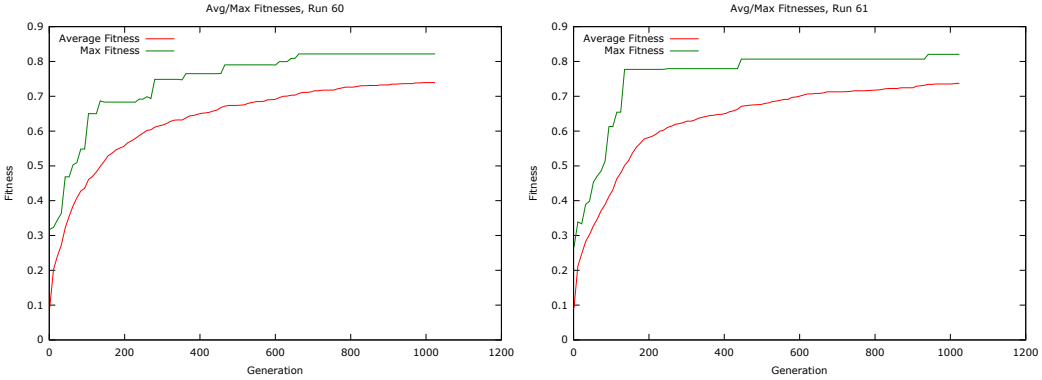

Figure 5.2: Fitness over time, runs 60 and 61

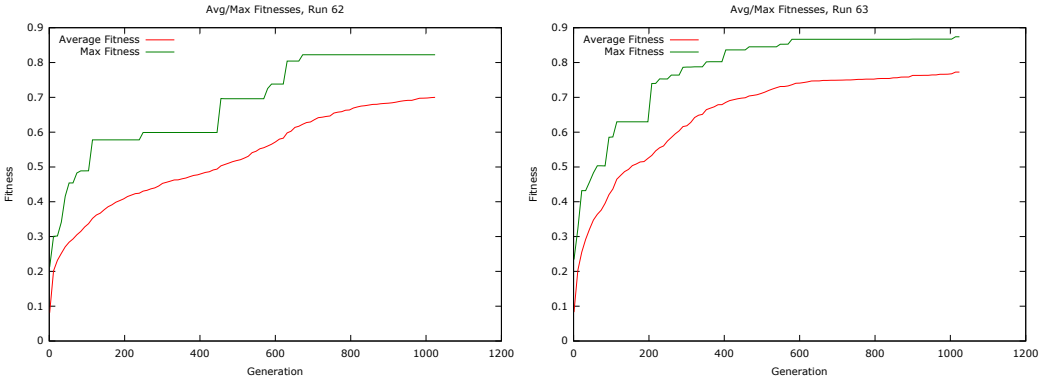

Figure 5.3: Fitness over time, runs 62 and 63

	d_{8s}	d_{16s}	k
58	0.7863m	1.6015m	0.9647
59	0.8311m	1.6732m	0.9867
60	0.8216m	1.6881m	0.9479
61	0.8206m	1.6272m	1.0172
62	0.8222m	1.6024m	1.0538
63	0.8740m	1.6930m	1.0672

In the end, let's examine the genomes of the individuals above. As mentioned before, there are two floating point numbers denoting respectively velocity and phase of a servo. Table below contains the values sorted by servo id vertically and different individuals horizontally.

	58	59	60	61	62	63
servo 0	1.0, 0.656	-1.0, 0.401	1.0, 0.026	1.0, 1.0	-1.0, 0.685	-1.0, 0.731
servo 1	1.0, 0.366	1.0, 0.695	-1.0, 0.321	1.0, 0.898	1.0, 0.041	-1.0, 0.194
servo 2	1.0, 0.379	-1.0, 1.0	1.0, 0.629	-1.0, 0.082	-1.0, 0.441	1.0, 0.508
servo 3	1.0, 0.959	1.0, 0.291	1.0, 0.991	1.0, 0.211	-1.0, 0.837	-1.0, 0.267
servo 4	-1.0, 0.629	1.0, 0.000	-1.0, 0.505	-1.0, 0.461	1.0, 0.000	-1.0, 0.320
servo 5	1.0, 1.0	-1.0, 0.686	-1.0, 0.000	-1.0, 0.910	-1.0, 0.163	-1.0, 0.055
servo 6	-1.0, 0.246	1.0, 0.472	1.0, 0.254	1.0, 0.423	-1.0, 0.092	-1.0, 0.340
servo 7	-1.0, 0.667	1.0, 0.918	1.0, 0.720	-1.0, 0.320	-1.0, 0.532	-1.0, 0.351

5.1.1 Evaluation

In 4 we've set 3 measures requiring examination with associated conditions. From the first table in 5.1 we can clearly see that all runs succeeded at evolving gaits achieving the set velocity of $5\frac{cm}{s}$ with a good margin. Figures 5.1, 5.2 and 5.3, especially from runs 62 and 63, suggest that chosen amount of generations or population was possibly not sufficiently large and that potentially better gaits could have been generated by the simulator. Based on acheved fitnesses, I estimate that the optimal movement speed in simulator should be between $0.90\frac{m}{8s} - 0.95\frac{m}{8s}$.

It is interesting to note how in all runs the best solution used maximum allowed velocity for the servos and used phases to control gaits. This implies that the velocity parameter is redundant and only increases complexity of evolution and time of convergence without affecting performance. Direction of the rotation does not in fact change anything - the same effect could be achieved be skewing phases. Another implication is that the gaits have a small period, reflected also by the value of calculated constant k, fluctuations of which are caused strictly by the effects of starting in a fully retracted position.

We can conclude that the simulator generated viable gaits and continue the experiment.

5.2 Hardware

The goal of this experiment is to validate that controllers generated by the simulating software can be transferred to reality. As described in 4.2, the inspection will happen visually. Figures 5.4 and 5.5 show comparison of robot's representation in simulation and the real robot. Note that some

of the images have a slightly different perspective, which might distort angles and lengths.

Additionally, all the solutions were run simultaneously in simulator and reality, stopped after 2s and 8s, and compared to each other. All the 6 individuals were tested, with results very similar to images from run 60 found in figures 5.6 and 5.7.

5.2.1 Evaluation

The similarity of starting positions is evident. There is, however, a difference in poses between simulation and reality in images showing starting positions for runs 61 and 63. One could speculate that the discrepancies are caused either by a defective model, imprecise measurements of the angles described in section 3.3 or by lack of robustness of the physical robot itself. The inaccuracies seem to be minor and affecting mostly positions approaching full retraction or extension of outer joints in the legs. While this problem does not seem to significantly disturb gaits, it should be examined in more details before more comprehensive experiments can be conducted.

5.3 Reality Gap and Transferability Approach

Using algorithm described in 3.4.4, 20 individuals were chosen for the initial transfer set:

run	individuals
58	0, 4, 14, 56
59	3, 41, 43, 45
60	3, 4, 7, 15, 27, 28, 43, 48, 57, 58
61	0, 9, 31

Each individual was tested once, so external factors could have skewed the results of the runs. Distances achieved in the tests are shown in a form of a boxplot in figure 5.8. As expected, the performance of the gaits in reality was much more variable than in simulator.

For each member of the transfer set a disparity value was calculated based on chosen characteristics. The values themselves are not necessarily interesting and their main goal is to be used as a basis for the SVM-based surrogate model. Once the vectors were input back into the simulator, two new simulation runs were made, runs 90 and 91, this time with minimizing disparity value as an objective, in addition to maximizing the movement distance. The results of the simulations are shown in the figure 5.9. There are clearly multiple non-dominated solutions, each of which could be chosen as the most fit. The first transfer set contained individuals with computed disparity values ranging from 0.345 to 1.0873. We can

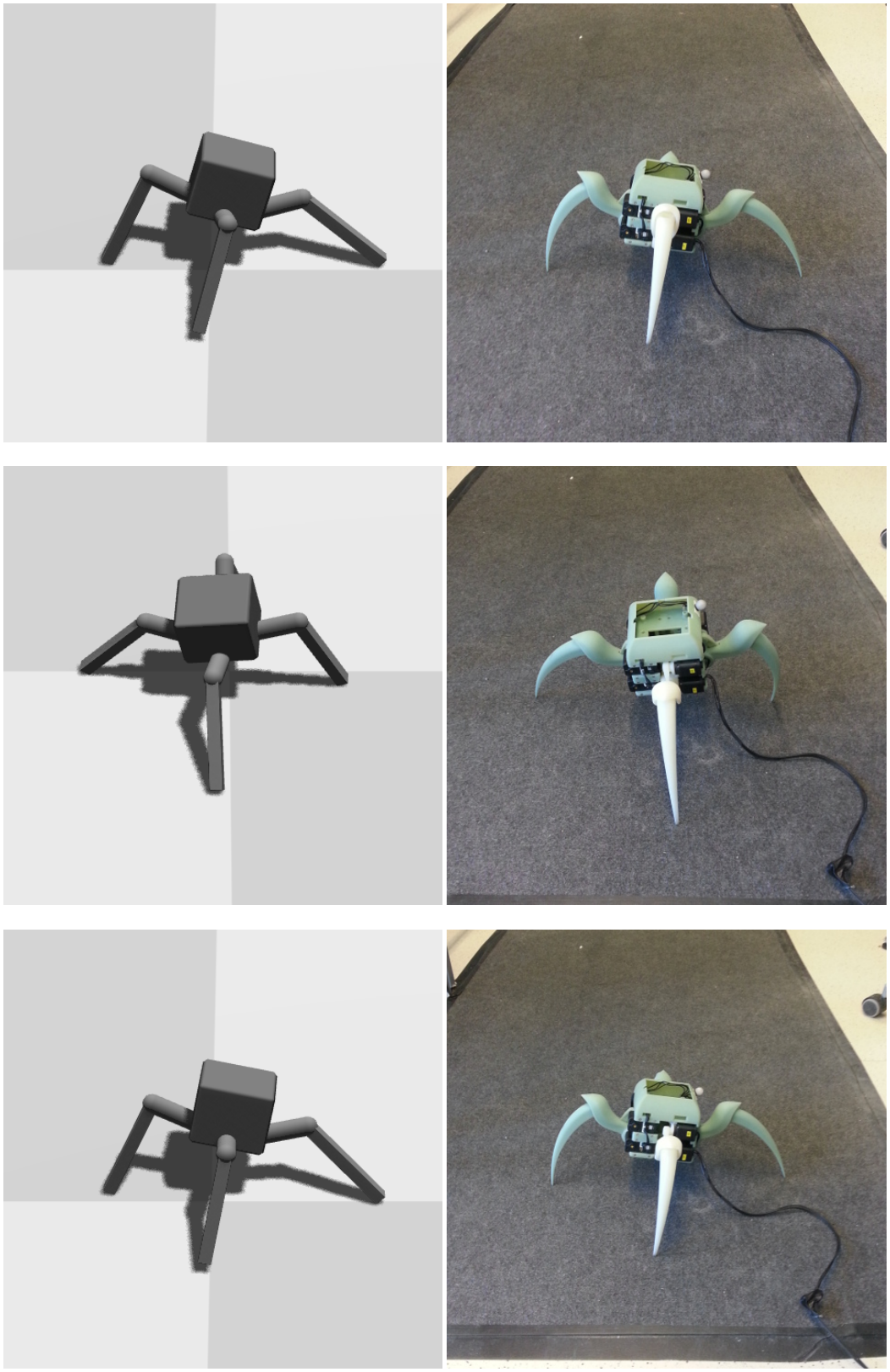

Figure 5.4: Comparison of starting positions from runs 58, 59 and 60

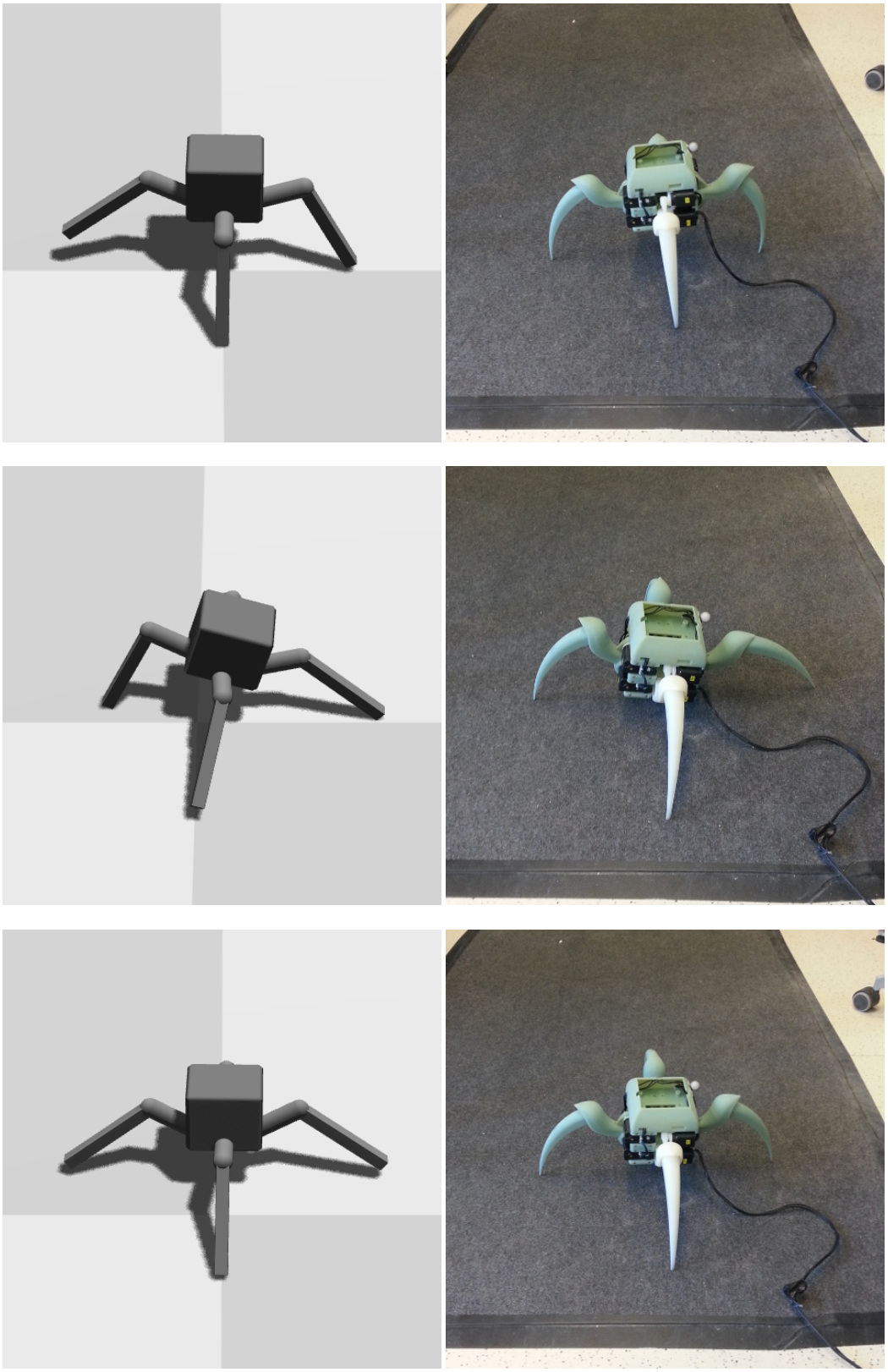

Figure 5.5: Comparison of starting positions from runs 61, 62 and 63

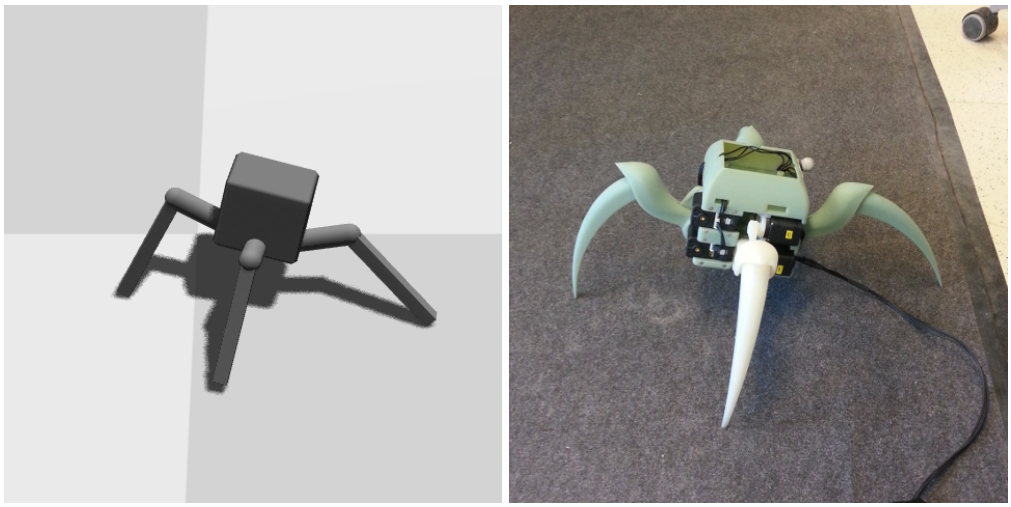

Figure 5.6: Position of the best individual from run 60 after 2 seconds

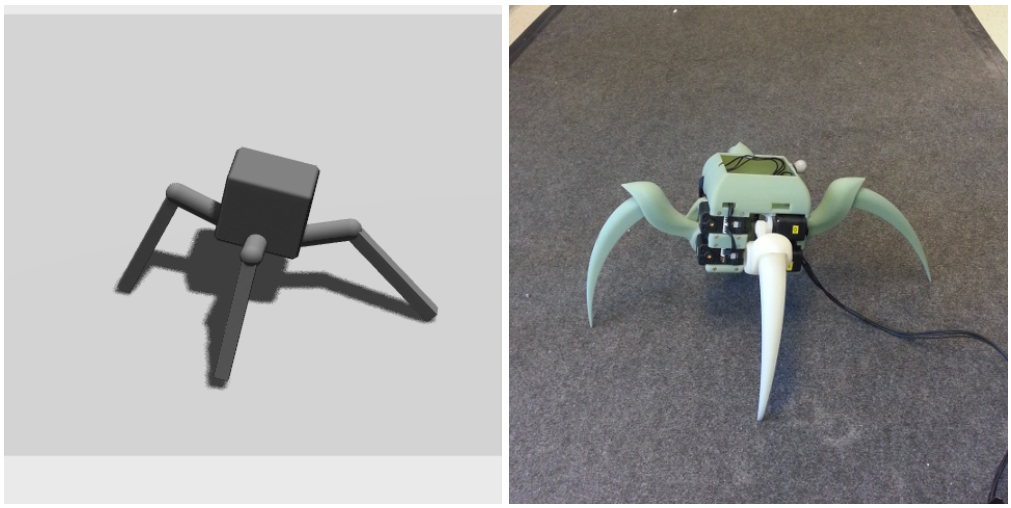

Figure 5.7: Position of the best individual from run 60 after 8 seconds

observe that all the new solutions have reliably low estimated disparity value (0.579 to 0.614) and a considerably lower predicted movement distance the the first 6 simulations. This was expected, as it is normal for genetic algorithms to use flaws of the simulator to create solutions that cannot work in the same way in reality, thus obtaining higher disparity values. Movement distance is one of the measures of transferability, so it is possible that the surrogate model learns that gaits scoring unusually high well in that dimension are not transferable.

From the second set of simulations, runs 90 and 91, another 10 individuals were chosen using the method from 3.4.4, now with the last 20 individuals already in population. They were tested in the exact same manner as the initial transfer set, and the spread of obtained distances can be seen in figure 5.10.

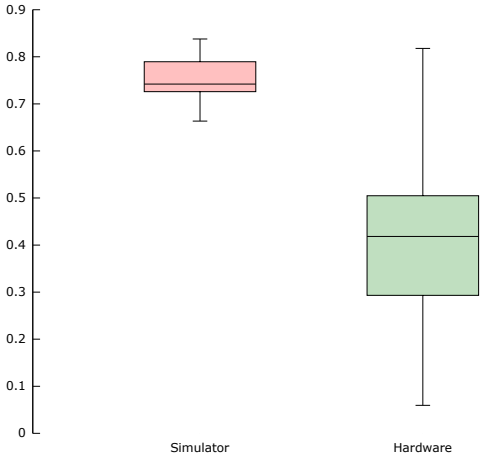

Figure 5.8: Distance achieved by individuals in the initial transfer set

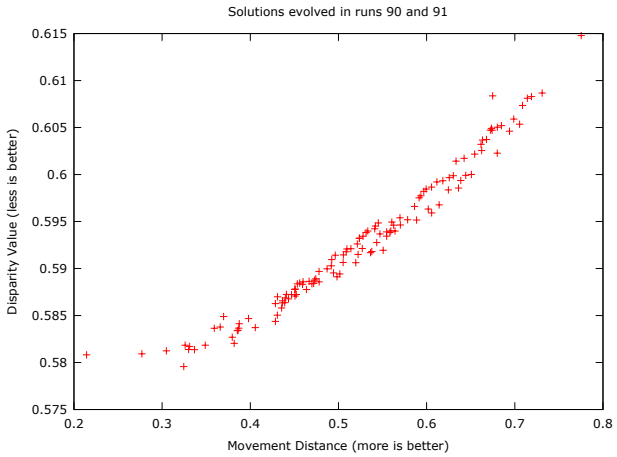

Figure 5.9: Individuals evolved in runs 90 and 91; with better fitness the closer they are to the bottom right corner.

One could think it is surprising for the individuals to have both bigger spread and lower mean and maximal travelled distance, but we have to remember that they were chosen based on their diversity with regard to the rest of the set of transferred controllers $\mathfrak{C}_T$. To assess how success of the transfer experiment, we have to look at data in other way. And indeed, a simple comparison of simulated distance $\hat{d}$ against distance on the real robot d, lined up with predicted disparity value $\hat{D}^*$ and actual disparity value D^* reveal some interesting tendencies:

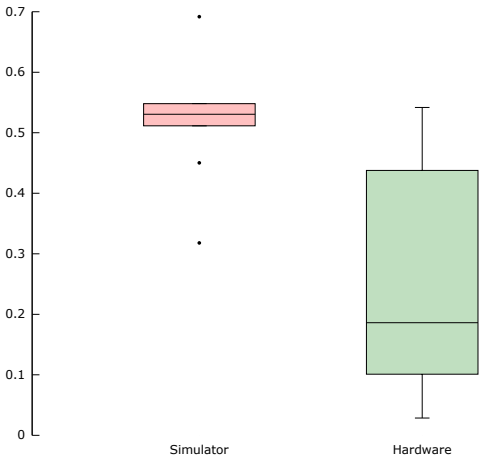

Figure 5.10: Distance achieved by individuals in the second transfer set

run-individual	$\hat{d}$	d	$\hat{D}^*$	D^*
90-50	0.45027	0.06066	0.58859	0.54953
91-52	0.51149	0.54189	0.59208	0.19676
90-15	0.52497	0.15979	0.59168	0.37583
90-61	0.31795	0.24372	0.58184	0.23483
91-56	0.69186	0.02854	0.60520	0.72262
90-7	0.54826	0.10096	0.59344	0.54946
91-2	0.53628	0.49755	0.59383	0.12904
90-53	0.53617	0.43778	0.59368	0.18098
90-2	0.54803	0.21255	0.59384	0.49431
90-16	0.51555	0.15836	0.59150	0.52871

Almost none of the individuals got a disparity value higher than the surrogate model predicted, except for 91-56. The individuals that transferred well with regard to movement distance (91-52, 91-2 and 90-53) also got very low disparity value. The last point is at least in part caused by including distance as a behaviour.

With that results, the last pair of simulations could be done (runs 92 and 93). Intuitively, the individuals should have even lower predicted disparity value, which in the end will hopefully result in smaller reality gap. And indeed, ignoring the single outlier, with a predicted disparity value of 0.564, the solutions are expected to transfer better to reality (fig. 5.11). The outlier is non-dominated because of its high movement in the simulator, indicating that movement distances greater than or equal are deemed not viable by the surrogate model. It is important to notice that the simulated movement distance range is about the same as in the last round of simulations, which implies that the resulting lower disparity value is based on parameters other than the distance. This is exactly the behaviour that is desired, and if other behaviours that were chosen are actually affecting transferability, then this should result in less reality gap

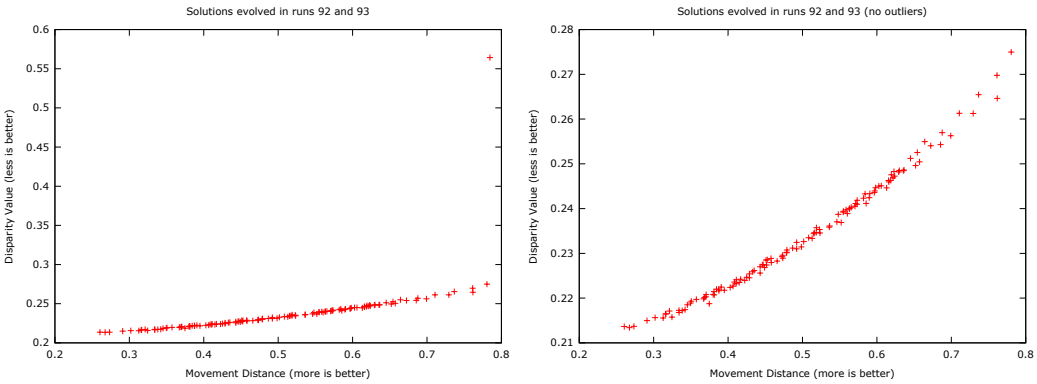

Figure 5.11: Individuals evolved in runs 92 and 93, respectively with and without the single outlier

in total.

As for choosing individuals to test in reality, I opted to use a semi-random approach. From each of the control sets (runs 62 and 63), the individual with the highest predicted movement distance was chosen and 9 were selected at random. Choosing the best individual from a result of multi-objective optimization is not necessarily as simple, but from each simulation run the ones with the highest movement (without outliers) and the ones with lowest estimated disparity value were chosen, in addition to 8 that were picked randomly.

In the end, the following individuals were used in the test:

run	individuals
62	0, 1, 12, 15, 17, 29, 37, 42, 48, 62
63	0, 8, 14, 18, 20, 25, 31, 32, 50, 58
92	11, 17, 19, 21, 37, 38, 44, 52, 57, 63
93	11, 19, 20, 25, 27, 29, 50, 52, 62, 63

Testing on the real robot did note yield results we were hoping to get. The comparison of distances in simulator against distances in reality can be found in figure 5.12. Except for a single outlier, they look rather similar to the ones achieved before. Using disparity value in evolution seemed to hinder movement both in simulations and in reality, thus not achieving the desired result. This effect can be seen even better in a direct comparison of results on robot seen in figure 5.13.

5.3.1 Evaluation

Reality gap proved to be a problem indeed - performance of generated gaits was significantly better in simulations than reality. There ware no obvious suggestions for reasons of the inaccuracy in how the real

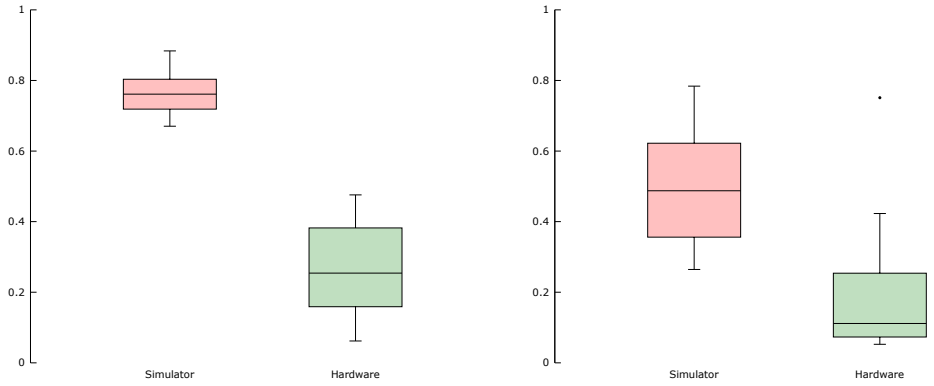

Figure 5.12: Transfer experiments with control sets vs the sets evolved using the disparity value calculation

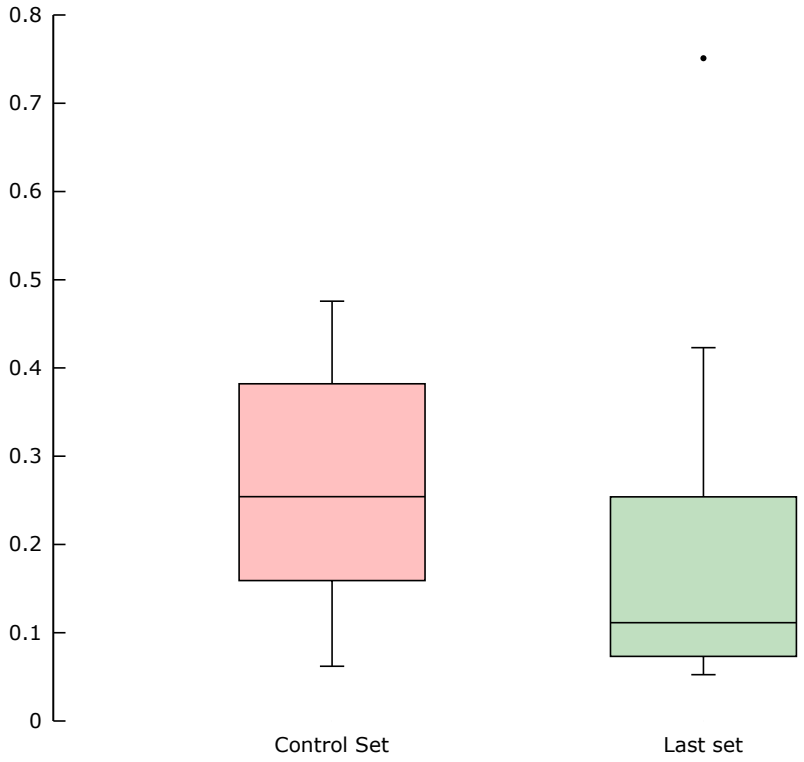

Figure 5.13: Comparison of results on hardware of individuals evolved without and with approximation of disparity value

robot performed. Despite running several transfer experiments, no improvement has been noted in the random sample of the controllers.

The gaits that performed best in reality had subjectively the most natural look in the simulator. There were two natural types of generated gaits - one that happened in the direction of one of the legs, and in the directions in between. Both had examples of successful transfers, thus validating that there are viable gaits for the robot and that they can be created in simulator.

Chapter 6

Discussion

One of the main goals for this thesis was to develop a framework for generating and testing gaits using Aracna. The results from the last chapter indicate that the simulator creates positions and walking patterns that are successfully transferred to reality. There is, however, despite the apparent similarity of behaviours in both environments, a considerable reality gap in how well the same gaits perform on the real robot in terms of movement distance.

There are many possible reasons for the reality gap. The model of the robot is vastly simplified with regards to structure, which in itself might induce a lot of difficulties. The approximation of the power transmission system from motors to joints is probably one of the main problems. While positions and velocities of the joints seem to be more or less correct, we lose all approximations of power and torque of the servos. The mechanical advantages of the system used in Aracna somewhat reduce the problem by offering joints a lot more torque than servos by themselves would.

Another major simplification is the point of contact with the ground. Value for friction in the simulator were chosen based on empirical testing, but the surface on which the robot walks in reality, a carpet, is slightly irregular and might cause the legs to get stuck or or slide more than expected. This effect is not easy to approximate in the simulator. Few of the gaits were relying on dragging side legs (relative to the direction of the movement) across the surface, which worked well in the simulator due to a small gap between tips of the legs and the floor.

Minor discrepancies discovered and shortly described in section **??** can be another reason for the reality gap. They seem to affect robot independently of position, possibly invalidating any attempts of avoiding gaits that transfer worse. Despite a detailed examination, no reason has been found for this behaviour. It is possible that the physical robot used in this thesis can in itself be the origin of the problem. The margins used for 3D-printing of legs caused some slack in the finished joints, which in addition to some wear could have decreased precision of the movements. Though improbable to cause that much reality gap, this should be taken

into consideration as well.

6.1 Transferability Approach

The second goal of this thesis was to apply the Transferability Approach to the evolution in order to minimize reality gap. The results of the transfer experiments of the population evolved with an approximation of disparity value were not impressive compared to the control set. A slight decline in movement distance was observed, without any apparent advantages. One could speculate that lower results in reality compared to the control set could have been caused by a lower achieved movement distance in the simulator, which happened because of an additional objective.

However, the results do not prove anything. The most probable reason for why the experiment didn't yield the expected results is that the choice of descriptors used to compute the disparity value, that is direction and distance of the movement and changes of orientation thorough the runs (described in 4.3), was not a good one. Due to a limited amount of data from transfer experiments, we don't want to have more than a few different descriptors at the same time, but in retrospect it can be said that another measures of transferability should have been tried at an earlier stage. The parameters were chosen based on subjective evaluation of earlier runs, which could have been not representative.

The amount of data on which the results were based on can not be called as significant. Populations in single runs had a tendency to converge on similar gaits with minor discrepancies between individuals. Two runs creating the control set and two runs using disparity value are not necessarily significant. With a different random number generator seeds the results could have been very different. In any case, more experiments are required.

6.2 Aracna

The last of the goals was to evaluate Aracna as a platform for generating controllers using Evolutionary Algorithms and applying the Transferability Approach. While some of the concepts used in the robot, such as a single-body leg design are interesting, they lead to few problems as well. At one point in experiments one of the legs broke and a new one had to be 3D-printed. The design of the joints made it impossible to get the support material out of all the gaps without breaking the leg. This made both joints in the leg immovable and therefore useless. Only after increasing the gaps between parts was it possible to remove the unnecessary material, and it came at the cost of some undesired slack in joints. A good design should optimally have neither of the problems.

The servo-to-joint transmission of torque used in Aracna creates a possibility for creating simple gaits by defining only relative phases between servos, but requires that the phases stay constant thorough a run. The necessary adjustments would be more simple to make if the servos used in the robot did not return false and at times random values for their current position $\frac{1}{6}$ of the time.

All of this adds to the reality gap of gaits generated for the robot. It is possible that local search techniques [51, 38] can somewhat mitigate the problem. A related problem is that small differences in genomes led to big differences in performance of the gaits, which would maybe be not as prominent if a wave-controller was used.

To summarize, the robot is an interesting platform for creating gaits, but due to a relatively small search space and big effects of reality gap not one that is simple to develop in simulations. Much better results could probably have been achieved by using local search optimization, or even global evolutionary algorithms directly on the hardware.

6.3 Future Work

As the time was rather scarce during creation of the framework, there is a lot to be done in the future. If we assume that using Aracna is a goal, the first and most important goal should be to improve its model in the simulator. As mentioned before (especially in 6), currently implemented representation of Aracna is not very detailed. More experiments should be made to determine cause of and reduce the reality gap described in section 5.2.1. This might be as simple as adding a constant offset to counter effects of slack in joints, but it remains to be tested.

The Transferability Approach did not yield any positive results in this thesis, but it does not follow that the idea or implementation was wrong. Parameters used for computing the disparity value - movement distance, movement direction and sum of differences of orientation - might have been incorrect choices. Other parameters should be tested. Additionally, there are some parameters that could not have been obtained due to the limited amount of information returned from the motion capture studio software. Statistics based on values of servos (current velocity and torque) or positions of joints could possibly be used in this context. It could also be interesting to attempt to apply Transferability Approach to robots using more traditional joints and with a greater range of possible gaits, like QuadraTot [21]. Another idea might be to use local search on gaits generated both with and without Transferability Approach. If the reality gap was initially too big and skewed the results of this experiment, then using the local search might create more reliable gates if they are created with their transferability as an objective.

As for Aracna, a clear limitation could be seen in the conducted

simulations in amount of different gaits that were generated. A greater search space could provide many interesting gaits and the simplest method of achieving that would be to use another controller. A simple wave controller could create gaits impossible with the current method. It is difficult to say how it would affect performance of the robot in simulation and reality, but it would most likely create a better environment for using the Transferability Approach.

6.4 Conclusion

The simulator created by the ROBIN-group at the University of Oslo was used for creating a model of Aracna and developing gaits using Multi-Objective Evolutionary Algorithms. In the process of creating the framework many parts of the simulator had to be rewritten or created. Implementation of the model and mapping from controller onto the real robot were not straightforward due to the limitations of the hardware. New solutions had to be created in order to be able to transfer movements precisely.

The framework was created, but despite apparent correctness of all the steps, the reality gap affected the performance of gaits by too big a factor to get any meaningful information from attempts on the real robot. Therefore a conclusion was made that while Aracna is an exciting platform for evolving gaits, it is non-trivial to conduct experiments in simulator and then transfer them to reality. A more direct approach to the hardware might be necessary to create optimal gaits.

The created framework was then extended to use Transferability Approach to evolve controllers predicted to transfer well to reality. First tests yielded no conclusive data with regard to how it affects the reality gap and more experiments are needed to determine its viability. Gaits created without using the concept have on average performed slightly better both in simulations and on the real robot. We can hypothesise that the Transferability Approach was used in a wrong manner and that behaviours chosen in the experiment were not representative of performance of the gaits. Either way more data is necessary to be able to make a definite conclusion.

The results of this thesis are thus a good starting point for a further set of experiments, both with regard to creating gaits for Aracna and applying Transferability Approach to other robots.

Bibliography

[1] Robotis e-manual v1.15.00 - ax-18f/ ax-18a.

[2] Torgny Brogårdh. Present and future robot control developmentâĂŤan industrial perspective. *Annual Reviews in Control*, 31(1):69–79, 2007.

[3] Sébastien Cahon, Nordine Melab, and E-G Talbi. Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357–380, 2004.

[4] Chang, Chih-Chung, Lin, and Chih-Jen. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[5] Vladimir Cherkassky and Yunqian Ma. Practical selection of svm parameters and noise estimation for svm regression. *Neural networks*, 17(1):113–126, 2004.

[6] Sonia Chernova and Manuela Veloso. An evolutionary approach to gait learning for four-legged robots. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2562–2567. IEEE, 2004.

[7] Dave Cliff, Phil Husbands, and Inman Harvey. Explorations in evolutionary robotics. *Adaptive behavior*, 2(1):73–110, 1993.

[8] Jeff Clune, Kenneth O Stanley, Robert T Pennock, and Charles Ofria. On the performance of indirect encoding across the continuum of regularity. *Evolutionary Computation, IEEE Transactions on*, 15(3):346–367, 2011.

[9] Carlos A Coello Coello and Gary B Lamont. *Applications of multi-objective evolutionary Algorithms*, volume 1. World Scientific, 2004.

[10] Ivo Couckuyt, Dirk Gorissen, Hamed Rouhani, Eric Laermans, and Tom Dhaene. Evolutionary regression modeling with active learning: An application to rainfall runoff modeling. In *Adaptive and Natural Computing Algorithms*, pages 548–558. Springer, 2009.

[11] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. Exploration and exploitation in evolutionary algorithms: a survey. *ACM Computing Surveys (CSUR)*, 45(3):35, 2013.

[12] James F Crow and Motoo Kimura. Efficiency of truncation selection. *Proceedings of the National Academy of Sciences*, 76(1):396–399, 1979.

[13] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[14] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of a*. *Journal of the ACM (JACM)*, 32(3):505–536, 1985.

[15] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[16] Agoston E Eiben, P-E Raue, and Zs Ruttkay. Genetic algorithms with multi-parent recombination. In *Parallel Problem Solving from Nature–PPSN III*, pages 78–87. Springer, 1994.

[17] Agoston E Eiben and James E Smith. *Introduction to evolutionary computing*. springer, 2003.

[18] Agoston Endre Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter control in evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on*, 3(2):124–141, 1999.

[19] Dario Floreano, Francesco Mondada, et al. Evolution of plastic neurocontrollers for situated agents. In *From animals to animats IV: proceedings of the fourth international conference on simulation of adaptive behavior*, volume 4, 1996.

[20] Carlos M Fonseca, Peter J Fleming, et al. Genetic algorithms for multiobjective optimization: Formulationdiscussion and generalization. In *ICGA*, volume 93, pages 416–423, 1993.

[21] Kyrre Glette, Gordon Klaus, Juan Cristobal Zagal, and Jim Torresen. Evolution of locomotion in a simulated quadruped robot and transferral to reality. In *Proceedings of the Seventeenth International Symposium on Artificial Life and Robotics*, 2012.

[22] David Edward Goldberg et al. *Genetic algorithms in search, optimization, and machine learning*, volume 412. Addison-wesley Reading Menlo Park, 1989.

[23] Twan Goosen, Rik van den Brule, Joris Janssen, and Pim Haselager. Interleaving simulated and physical environments improves evolution of robot control structures. In *Proceedings of the 19th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC)*, pages 135–142. Citeseer, 2007.

[24] Daniel H Grollman and Odest Chadwicke Jenkins. Learning robot soccer skills from demonstration. In *Development and Learning, 2007. ICDL 2007. IEEE 6th International Conference on*, pages 276–281. IEEE, 2007.

[25] Daniel Damir Harabor and Alban Grastien. Online graph pruning for pathfinding on grid maps. In *AAAI*, 2011.

[26] Robert Hinterding. Gaussian mutation and self-adaption for numeric genetic algorithms. In *Evolutionary Computation, 1995., IEEE International Conference on*, volume 1, page 384. IEEE, 1995.

[27] Yu-Chi Ho and David L Pepyne. Simple explanation of the no-free-lunch theorem and its implications. *Journal of Optimization Theory and Applications*, 115(3):549–570, 2002.

[28] Jeffrey Horn, Nicholas Nafpliotis, and David E Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 82–87. Ieee, 1994.

[29] Gregory S Hornby, Seiichi Takamura, Takashi Yamamoto, and Masahiro Fujita. Autonomous evolution of dynamic gaits with two quadruped robots. *Robotics, IEEE Transactions on*, 21(3):402–410, 2005.

[30] Jens Jägersküpper and Tobias Storch. How comma selection helps with the escape from local optima. In *Parallel Problem Solving from Nature-PPSN IX*, pages 52–61. Springer, 2006.

[31] Nick Jakobi. *Minimal simulations for evolutionary robotics*. PhD thesis, University of Sussex, 1998.

[32] Nick Jakobi, Phil Husbands, and Inman Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In *Advances in artificial life*, pages 704–720. Springer, 1995.

[33] J-C Jouhaud, P Sagaut, M Montagnac, and J Laurenceau. A surrogate-model based multidisciplinary shape optimization method with application to a 2d subsonic airfoil. *Computers & Fluids*, 36(3):520–529, 2007.

[34] Cory D Kidd, Will Taggart, and Sherry Turkle. A sociable robot to encourage social interaction among the elderly. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 3972–3976. IEEE, 2006.

[35] Joshua Knowles and David Corne. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 1. IEEE, 1999.

[36] Sylvain Koos, J-B Mouret, and Stéphane Doncieux. The transferability approach: Crossing the reality gap in evolutionary robotics. *Evolutionary Computation, IEEE Transactions on*, 17(1):122–145, 2013.

[37] Slawomir Koziel and John W Bandler. Space-mapping optimization with adaptive surrogate model. *Microwave Theory and Techniques, IEEE Transactions on*, 55(3):541–547, 2007.

[38] Jeffrey C Lagarias, James A Reeds, Margaret H Wright, and Paul E Wright. Convergence properties of the nelder–mead simplex method in low dimensions. *SIAM Journal on Optimization*, 9(1):112–147, 1998.

[39] Sara Lohmann, Jason Yosinski, Eric Gold, Jeff Clune, Jeremy Blum, and Hod Lipson. Aracna: An open-source quadruped platform for evolutionary robotics. In *Artificial Life*, volume 13, pages 387–392, 2012.

[40] Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[41] Franklin Mendoza, Jose L Bernal-Agustin, and Jose A Dominguez-Navarro. Nsga and spea applied to multiobjective design of power distribution systems. *Power Systems, IEEE Transactions on*, 21(4):1938–1945, 2006.

[42] Brad L Miller and David E Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9(3):193–212, 1995.

[43] Brad L Miller and David E Goldberg. Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, 4(2):113–131, 1996.

[44] Jean-Baptiste Mouret, Sylvain Koos, and Stéphane Doncieux. Crossing the reality gap: a short introduction to the transferability approach. *arXiv preprint arXiv:1307.1870*, 2013.

[45] Stefano Nolfi and Dario Floreano. *Evolutionary robotics*, volume 150. MIT press Cambridge, 2000.

[46] Aidan O'Dwyer. *Handbook of PI and PID controller tuning rules*, volume 2. World Scientific, 2009.

[47] Riccardo Poli and William B Langdon. On the search properties of different crossover operators in genetic programming. *Genetic Programming*, pages 293–301, 1998.

[48] Marc Raibert, Kevin Blankespoor, Gabriel Nelson, Rob Playter, et al. Bigdog, the rough-terrain quadruped robot. In *Proceedings of the 17th World Congress*, pages 10823–10825, 2008.

[49] Thomas Röfer. Evolutionary gait-optimization using a fitness function based on proprioception. In *RoboCup 2004: Robot Soccer World Cup VIII*, pages 310–322. Springer, 2005.

[50] Karl Sims. Evolving 3d morphology and behavior by competition. *Artificial life*, 1(4):353–372, 1994.

[51] James C Spall. An overview of the simultaneous perturbation method for efficient optimization. *Johns Hopkins APL Technical Digest*, 19(4):482–492, 1998.

[52] William M Spears and Kenneth D De Jong. On the virtues of parameterized uniform crossover. Technical report, DTIC Document, 1995.

[53] Russell H Taylor and Dan Stoianovici. Medical robotics in computer-integrated surgery. *Robotics and Automation, IEEE Transactions on*, 19(5):765–781, 2003.

[54] Dirk Thierens and David Goldberg. Elitist recombination: An integrated selection recombination ga. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 508–512. IEEE, 1994.

[55] Ethan Tira-Thompson. Digital servo calibration and modeling. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-09-41*, 2009.

[56] Euskal Herriko Unibertsitatea and Informatika Fakultatea. On why better robots make it harder. In *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, volume 3, page 64. MIT Press, 1994.

[57] Welch and William J. Algorithmic complexity: three np- hard problems in computational statistics. *Journal of Statistical Computation and Simulation*, 15(1):17–25, 1982.

[58] David Wettergreen and Chuck Thorpe. Gait generation for legged robots. In *IEEE International Conference on Intelligent Robots and Systems*, 1992.

[59] Gerhard J Woeginger. Exact algorithms for np-hard problems: A survey. In *Combinatorial OptimizationâĂŤEureka, You Shrink!*, pages 185–207. Springer, 2003.

[60] Tianfang Xu, Albert J Valocchi, Jaesik Choi, and Eyal Amir. Improving groundwater flow model prediction using complementary data-driven models. In *XIX International Conference on Computational Methods in Water Resources, Univ. of Ill., Urbana-Champaign, Ill*, 2012.

[61] J.C. Zagal, J. Ruiz-del-Solar, and P. Vallejos. Back-to-Reality: Crossing the reality gap in evolutionary robotics. In *IAV 2004: Proceedings 5th IFAC Symposium on Intelligent Autonomous Vehicles*. Elsevier Science Publishers B.V., 2004.

[62] Ting Zhu et al. Application of surrogate modeling to generate compact and pvt-sensitive ibis models. In *2009 IEEE 18th Conference on Electrical Performance of Electronic Packaging and Systems*, pages 77–80, 2009.

[63] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195, 2000.

[64] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm, 2001.

[65] Viktor Zykov, Josh Bongard, and Hod Lipson. Evolving dynamic gaits on a physical robot. In *Proceedings of Genetic and Evolutionary Computation Conference, Late Breaking Paper, GECCO*, volume 4, 2004.